Python : débogueur, bibliothèques, objets

Table des matières

1 Utilisation d'un débogueur

Les boutons à utiliser sont ici nommés comme dans l'IDE *Spyder*, mais le lecteur n'aura aucun mal à retrouver des commandes similaires dans n'importe quel autre débogueur.

1.1 Suivi de variables

Ouvrir l'*explorateur de variables* et exécuter le programme avec la commande *Déboguer* du menu *Déboguer* (raccourci *Ctrl+F5* ou ⏯).

Pour exécuter la ligne du programme surlignée, cliquer sur le bouton *Exécuter la ligne en cours* ↷, ou bien taper `n` (comme `next`) dans l'interpréteur.

1.2 Points d'arrêt

Se placer sur la ligne à laquelle on désire placer un point d'arrêt ● et choisir *Ajouter un point d'arrêt* dans le menu *Déboguer* (raccourci *F12*), ou encore double-cliquer dans la marge gauche du programme.

Alors *Continuer l'exécution jusqu'au prochain point d'arrêt* (⏩ ou `c` dans l'interpréteur interactif) exécute le programme normalement jusqu'au prochain point d'arrêt.

EXERCICE 1 *1. Taper le programme suivant dans l'éditeur et l'exécuter. Que se passe-t-il ?*

```
i = 10
while i != 0:
    i = 1-i
    print(i)
```

(Il est possible qu'à ce stade il soit nécessaire de fermer complètement l'IDE et de le rouvrir pour continuer...).

2. *Exécuter ce programme pas à pas et observer les valeurs successives prises par la variable* `i`*. Expliquer le comportement observé à la première question.*
3. *Placer un point d'arrêt à un endroit approprié du programme pour montrer son comportement sans avoir besoin de détailler des étapes inutiles.*

EXERCICE 2 *Utiliser un débogueur pour identifier l'erreur commise dans le programme ci-après. Le comportement attendu est le suivant : on demande des nombres entiers au clavier tant que la suite des nombres fournis est strictement croissante. La variable* `n` *compte le nombre de valeurs entrées au clavier.*

```
precedent = int(input())
nouveau = int(input())
n = 2
while precedent < nouveau:
    nouveau = int(input())
    precedent = nouveau
    n = n + 1
```

1.3 Entrer dans les fonctions (ou pas)

Lorsque le programme comporte des fonctions, considère chaque appel de fonction comme une instruction unique et passe donc directement à l'instruction suivante.

Pour entrer dans le corps d'une fonction au moment où elle est appelée, cliquer sur *Pas vers l'intérieur de la fonction* ou taper `s` comme `step`.

Si le débogueur est entré dans une fonction et pour aller directement à la fin de l'exécution de cette fonction, cliquer sur *Exécuter jusqu'au retour de la fonction* ou taper `r` comme `return`.

EXERCICE 3 *On donne le programme suivant :*

```
def f(x):
    for i in range(100):
        x = (13*x + 1) % 256
    return x

def g():
    s = 0.
    for j in range(10):
        a = f(j)
        s = s + 1./(a-210)
    return s

print(g())
```

1. *Que se passe-t-il lors de son exécution ?*
2. *Jusqu'à quel point la simple lecture du code permet-elle d'expliquer ce comportement ?*
3. *À quelle itération de la boucle de la fonction* `g` *se situe le problème ?*

Dans une fonction récursive, à chaque appel récursif, il est créé un nouvel ensemble de variables locales à la fonction. L'explorateur de variables ne montre que les variables en cours d'utilisation, autrement dit celles qui sont propres à l'appel dans lequel on se situe.

Taper `w` (pour `where`) dans l'interpréteur, on voit apparaître la liste des appels de fonctions en cours, le plus récent étant situé tout en bas. Ensuite, il est possible de se déplacer dans cette liste avec les commandes `u` (pour `up`) et `d` (pour `down`). L'explorateur de variables montre alors les contenus des variables correspondant à ces différents appels.

Wing IDE 101 propose une interface graphique pour effectuer ces manipulations, mais ce n'est pour l'instant pas le cas de Spyder.

2 Bibliothèques

Le contexte

Il s'agit de découvrir les différents outils proposés par Python pour l'ingénierie numérique. Au casting, nous trouvons comme différents intervenants :

- La librairie `numpy` fournit ce qu'il faut pour créer et manipuler des *tableaux* (`array`), qui sont des listes homogènes (que des entiers, que des flottants... mais surtout pas de mélange). La sous-librairie `numpy.linalg` contient des fonctions spécialisées d'algèbre linéaire.
- Au dessus, on trouve `scipy`, qui est une librairie de calcul scientifique basée sur `numpy`. Sa langue naturelle est le tableau.
- La librairie `matplotlib` fournit un ensemble de fonctions permettant de représenter toutes sortes de graphiques. Nous utiliserons essentiellement la sous-librairie `matplotlib.pyplot`.
- La fonction `odeint` de la sous-librairie `scipy.integrate` réalise des résolutions numériques d'équations différentielles.

Il ne s'agit pas d'entrer dans les détails des nombreuses options (la librairie `matplotlib` est en particulier très fournie !), mais plutôt de savoir faire assez rapidement et simplement des choses assez basiques. Les détails viendront avec la pratique et à l'aide de la documentation.

2.1 La librairie `numpy`

Un petit aide-mémoire pour commencer :

Quoi	**Comment**
Créer une matrice particulière	`zeros((n, p))` ; `random.random((n, p))` `eye(n)` (identité) ; `fromfunction(f, (n, p))`
Manipuler des matrices	`2*a` ; `a+b` ; `a.transpose()` ou `a.T` ; `dot(a, b)`
Sous-bibliothèque `numpy.linalg`	`inv(a)` ; `solve(a, b)` ; `eigvals(a)`

EXERCICE 4 *Écrire une fonction qui prend pour argument un entier n et qui crée la matrice tridiagonale de dimension n dont tous les coefficients valent 2 et dont tous les autres coefficients non nuls valent -1.*

EXERCICE 5 *Déterminer les valeurs propres d'une matrice A quelconque à l'aide de `numpy.linalg.eigvals`, puis les utiliser pour construire son polynôme caractéristique et l'appliquer à A. Retrouve-t-on le théorème de Cayley-Hamilton ?*

EXERCICE 6
1. *Programmer une fonction qui calcule le déterminant d'une matrice par développement selon une ligne ou une colonne.*
2. *Déterminer la complexité d'une telle procédure.*
3. *Vérifier cette complexité empiriquement.*
4. *Comment faire mieux ?*

2.2 La librairie `matplotlib`

La première chose à comprendre est que par défaut, `plot` va relier des points (donnés par listes ou tableaux d'abscisses et d'ordonnées). Et une courbe, c'est par définition une ligne brisée avec très exactement *plein* de points.

Quoi	Comment
Charger la librairie !	`import matplotlib.pyplot as pypl`
Tracer une ligne brisée	`pypl.plot([x1,...,xn], [y1...,yn])`

Mettre des titres/noms aux axes	`pypl.title('Titre de la figure')`
	`pypl.xlabel('Nom de l'axe des x')`
	`pypl.ylabel('Nom de l'axe des y')`

Visualiser le résultat	`pypl.show()`
Effacer la fenètre graphique courante	`pypl.clf()`
Sauver la figure (sous différents formats !)	`pypl.save('jolie-figure.bmp')` `pypl.savefig('jolie-figure.pdf')`

Voici un premier exemple de figure :

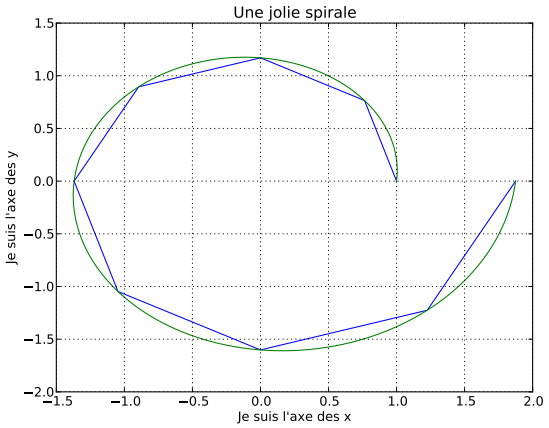

FIGURE 1 – Une spirale logarithmique

On peut l'obtenir avec les commandes suivantes :

Programme 1 — Un premier exemple

```
tt = np.linspace(0, math.pi*2, 9)
tt2 = np.linspace(0, math.pi*2, 1000)

def fx(t): return math.cos(t) * math.exp(t/10)
def fy(t): return math.sin(t) * math.exp(t/10)

pypl.plot(list(map(fx, tt)), list(map(fy, tt)))
pypl.plot(list(map(fx, tt2)), list(map(fy, tt2)))

pypl.title('Une␣jolie␣spirale')
pypl.xlabel('Je␣suis␣l\'axe␣des␣x')
pypl.ylabel('Je␣suis␣l\'axe␣des␣y')
pypl.grid(True)

pypl.savefig('FilesOut/spirale.pdf')
pypl.show()
```

EXERCICE 7 *Tracer une ellipse de rayons a et b dont les axes sont ceux du repère de trois façons différentes :*

- *en utilisant ses équations paramétriques*

– *en utilisant son équation cartésienne*
– *en utilisant son équation polaire (consulter la documentation de* `matplotlib`*...)*

Notons qu'il existe différentes solutions pour discrétiser un intervalle, plus ou moins agréables selon qu'on a fixé le pas, les bornes de l'intervalle ou encore le nombre de morceaux souhaités.

Programme 2 — Trois façons de casser l'intervalle $[1, 2]$ en dix morceaux.

```
>>> lp0 = [1 + float(k)/10 for k in range(11)]
[1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0]
>>> lp1 = np.arange(1, 2.1, 0.1)
array([ 1., 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. ])
>>> lp2 = np.linspace(1,2,11)
array([ 1., 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. ])
```

Pour le premier, on comprend bien ce qui se passe. Pour `arange` ça reste dans l'esprit Python (frontière droite exclue) et pour `linspace`, on compte les *poteaux* et non les *intervalles*.

2.3 La librairie `scipy`

Ce premier exercice ne concerne pas directement `scipy`, mais il montre une manipulation bien utile permise par Python.

EXERCICE 8 *1. Écrire une fonction Python permettant de calculer une approximation du nombre dérivé d'une fonction $f : \mathbb{R} \to \mathbb{R}$ en une abscisse x donnée.*

2. Modifier cette fonction pour obtenir un opérateur `derivee` *qui prend pour seul argument une fonction f et renvoie (une approximation de) f'.*

3. Appliquer la fonction `scipy.optimize.newton` *à une fonction f de votre choix :*
   – *une fois en fournissant votre approximation "maison" de f' ;*
   – *une autre fois en ne fournissant pas de dérivée (c'est alors la méthode de la sécante qui est utilisée).*

   Comparer les résultats obtenus (on pourra notamment s'intéresser à la précision obtenue pour un nombre d'itérations borné par le paramètre `maxiter`*).*

EXERCICE 9

Appliquer la méthode de Newton à la fonction $x \mapsto x^2 - K$ afin d'obtenir une approximation de $\sqrt{K}$.

Tester en particulier en prenant pour K le complexe $1 + i$ et différentes valeurs initiales x_0. Le complexe $1 + i$ est construit en Python par `complex(1, 1)` *: aucune bibliothèque n'est requise.*

Comparer le résultat à celui renvoyé par `complex(1, 1) ** 0.5`*.*

3 Intégrer une équation différentielle avec `odeint`

Le principe d'utilisation de `odeint` (pour intégrer numériquement des équations différentielles) est le suivant : pour avoir une estimation numérique de la solution du problème de Cauchy $\begin{cases} Y'(t) = F(Y(t), t) \\ Y(t_0) = Y_0 \end{cases}$ on donne comme argument la *fonction* F (qui `doit` avoir deux paramètres, même dans le cas automome, avec t (suivant les notations précédentes) comme *deuxième* paramètre), la condition initiale Y_0, et le domaine de temps qui nous intéresse (qui commence à t_0). Il nous est retourné un *tableau* (même si t était une liste).

Programme 3 — Une équation pas trop compliquée

```
from scipy.integrate import odeint

def f(y, _): return y

t = np.linspace(0, 1, 3)
y = odeint(f, 1, t)
pypl.plot(t, y)

t = np.linspace(0, 1, 100)
y = odeint(f, 1, t)
pypl.plot(t, y)

pypl.plot(t, np.exp(t))
```

Après le premier appel à `odeint`, le tableau `y` vaut :

```
array([[ 1.        ], [ 1.64872127], [ 2.71828191]])
```

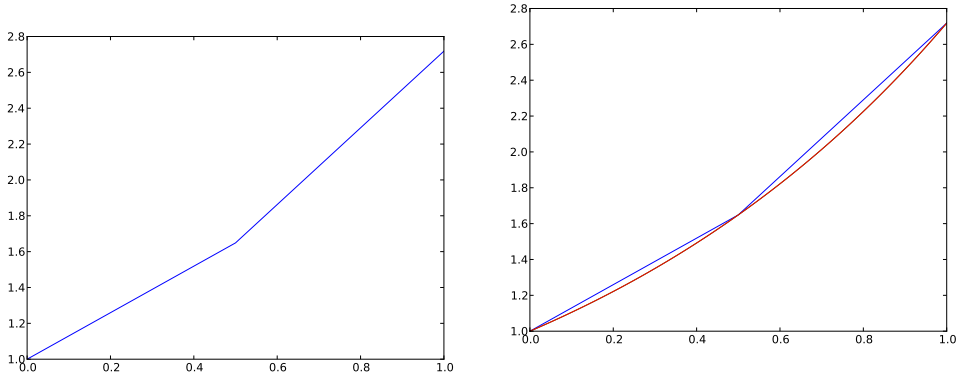

FIGURE 2 – Le résultat avec 3 points puis un peu plus

On notera que le nombre de points en lesquels les résultats sont évalués n'est pas (du moins directement) relié à la précision des calculs internes (ne pas imaginer que cela fixe le pas de la méthode, en particulier). Sur la seconde figure, la solution réelle se superpose à celle représentée en 100 points.

On peut également traiter le cas où Y est un k-uplet. Cela permet de traiter le cas d'un système différentiel du type $\begin{cases} y_1'(t) = F_1(y_1(t), y_2(t), t) \\ y_2'(t) = F_1(y_1(t), y_2(t), t) \end{cases}$ ce qui inclut les équations différentielles d'ordre 2 : la résolution de $y''(t) = f(y(t), y'(t), t)$ passera par celle du système différentiel $\begin{cases} y_1'(t) = y_2(t) \\ y_2'(t) = f(y_1(t), y_2(t), t) \end{cases}$

Dans le résultat (qui est une liste de couple, notons-la `res`), les premières composantes (que l'on récupère à la `numpy` via `res[:, 0]`) représentent les valeurs prises par y, et la seconde composante représentera celles de y'. Cela permettra de représenter au choix le graphe de y ($y(t)$ en fonction de t) ou bien le portrait de phase (les couples $(y(t), y'(t))$).

Le premier exemple est extrêmement original...

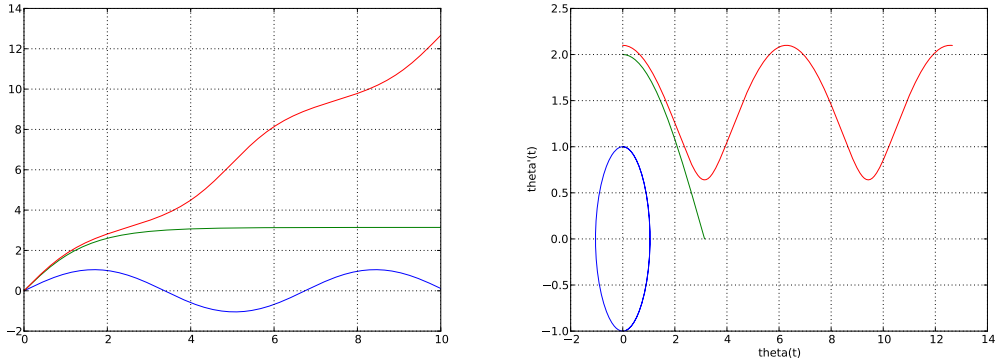

FIGURE 3 – Mais de quelle équation parle-t-on ?

Programme 4 — Le pendule non amorti

```
def pendule(igrec, t):
    [y, yp] = igrec # pratique, non ?
    return [yp, -math.sin(y)]

t = np.linspace(0, 10, 1000)
y1 = odeint(pendule, [0, 1], t)
y2 = odeint(pendule, [0, 2], t)
y3 = odeint(pendule, [0, 2.1], t)
pypl.plot(t, y1[:, 0])
pypl.plot(t, y2[:, 0])
pypl.plot(t, y3[:, 0])

pypl.grid()
pypl.show()
pypl.savefig('FilesOut/non-amorti.pdf')

pypl.clf()
pypl.grid()
pypl.plot(y1[:, 0], y1[:, 1])
pypl.plot(y2[:, 0], y2[:, 1])
pypl.plot(y3[:, 0], y3[:, 1])

pypl.savefig('FilesOut/non-amorti-phase.pdf')
pypl.show()
```

3.1 Exercices pour le matheux

EXERCICE 10 (ÇA TOURNE !) *Représenter les solutions des systèmes autonomes suivants, issues de* $(1,0)$ *:*

$$\begin{cases} x' &= -y - x(x^2+y^2) \\ y' &= x - y(x^2+y^2) \end{cases} \quad \begin{cases} x' &= -y - \frac{x}{10}(x^2+y^2) \\ y' &= x + \frac{y}{10}(x^2+y^2) \end{cases} \quad \begin{cases} x' &= -y + \frac{x}{20}(x^2+y^2) \\ y' &= x + \frac{y}{20}(x^2+y^2) \end{cases}$$

EXERCICE 11 (UNE INSTABILITÉ NUMÉRIQUE CLASSIQUE) *On s'intéresse ici à l'équation différentielle* $y'' = -y' + 6y$ *avec les conditions initiales :* $y(0) = 2$ *et* $y'(0) = -6$.

1. *Représenter la solution sur l'intervalle* $[0, 10]$.

2. *Calculer dans un coin, prendre un air intelligent, et expliquer que tout se passe comme prévu.*
3. *Réfléchir un peu, prendre un air inquiet, et expliquer que tout cela est anormal.* Indication : lire le titre !
4. *Réfléchir un peu plus (ou expérimenter), prendre un air ravi, pointer et expliquer le phénomène attendu !*
5. *Bonus : en déduire le nombre approximatif*[1] *de bits significatifs (bref : la taille de la mantisse) dans les flottants manipulés par Python.*

EXERCICE 12 (UNE ÉQUATION NON LINÉAIRE) *Représenter les graphes des solutions de l'équation* $y' = x^3 - y^3$ *avec différentes conditions initiales, par exemple de la forme* $y(-1) = \cdots$

3.2 Exercices pour le physicien

EXERCICE 13 (CHUTE (DE POMME)) *Si on lance un objet avec une vitesse (scalaire) fixe et différents angles, l'enveloppe des trajectoires est une parabole*[2]*.*

1. *Observer le phénomène, en représentant les solutions de* $z'' = -g$*, avec les conditions initiales* $(x(0), z(0)) = (0, 0)$*, et* $(x'(0), z'(0)) = (v_0 \cos\alpha, v_0 \sin\alpha)$ *pour différentes valeurs de* α*.*
   Pour ne représenter que les points à cote positive, j'ai filtré les valeurs à représenter (initialement dans `values`) de la façon suivante :
   ```
   valp = np.array([v for v in values if v[1]>=0])
   ```
2. *On suppose maintenant que le mouvement est soumis à une force de frottement :* $m\overrightarrow{a} = \overrightarrow{g} - k\overrightarrow{v}$*. Observer les différentes enveloppes obtenues (en faisant varier* k *: pour* $k = 0$*, on retrouve la parabole de sécurité).*
3. Pour les matheux égarés par ici : *trouver l'équation de l'enveloppe !*

EXERCICE 14 (CHUTE (DE GLAÇON)) *Si un glaçon tombe du haut d'un igloo, il va «décrocher» après un certain temps. Plus précisément, si on observe la réaction de l'igloo sur le glaçon, le décrochement*[3] *a lieu lorsque cette force s'annule.*

Si mes calculs sont bons[4]*, et en utilisant la conservation de l'énergie totale, on trouve un angle de décrochement de* $\arccos\frac{2}{3} \simeq 0.84$ *(j'ai noté* θ *l'angle entre* $\overrightarrow{OM}$ *et la verticale,* O *étant le centre de l'igloo et* M *la position du glaçon).*

1. *Sortir un crayon, trouver l'équation différentielle vérifiée par l'angle* θ *représentant ce qu'on imagine !* Mettre en particulier en avant la condition de décrochement.
2. *«Représenter la solution» pour différentes valeurs de* $\dot\theta(0) > 0$ *(il faut bien donner une impulsion !).*
   La conservation de l'énergie me donne $\frac{2}{3}\cos\theta = 1 + \frac{R}{2g}\dot\theta(0)^2$ au moment du décrochement.
3. *On suppose maintenant que le glaçon est soumis à un frottement solide :*
$$m\begin{pmatrix} r\ddot\theta \\ -r\dot\theta^2 \end{pmatrix} = mg\begin{pmatrix} \sin\theta \\ -\cos\theta \end{pmatrix} + \begin{pmatrix} -\varepsilon kT \\ T \end{pmatrix},$$
   avec $\varepsilon \in \{-1, 1\}$ *le signe de* $\dot\theta$*, et ceci tant que* $g\cos\theta > r\dot\theta^2$*.*
   À conditions initiales fixées, observer l'angle de décrochement en fonction de k*.*
   Pour k trop gros, le glaçon ne chutera finalement pas...

1. Allez, à 5 près !
2. Souvenirs de terminale en ce qui me concerne !
3. Je dirais plutôt décrochage, mais il semblerait que ce soit décrochement...
4. Hum... je les ai fait vérifier, donc ils semblerait qu'ils le soient !

4 Manipulations de fichiers

Débuter dans de bonnes conditions

Dans le répertoire de travail, créer des dossiers `FilesIn` et `FilesOut`. Récupérer et décompresser l'archive `https://ensiwiki.ensimag.fr/images/1/10/Materiel-tp-fichiers.zip`, puis placer dans `FilesIn/` les fichiers suivants :

`premierspremiers.txt`, `sudoku.txt` et `joconde.bmp`

Pour ceux qui sont joueurs, on peut aller chercher l'archive sur le web sans clic (tellement vulgaire...) avec la librairie `urllib`. De même la décompression d'un fichier `.zip` est prise en charge par la librairie `zipfile` (ça fonctionne au moins sous linux).

4.1 Lecture et écriture

Quoi	Comment
Ouverture en lecture	`monfichier = open('lenom.txt', 'r')`
Ouverture en écriture	`monfichier = open('lenom.txt', 'w')`
Fermeture	`monfichier.close()`
Lire la ligne suivante	`monfichier.readline()`
Lire toutes les lignes	`for l in monfichier.readlines():` `  ...`
Écrire dans le fichier... et passer à la ligne suivante	`monfichier.write('plof')` `monfichier.write('plof\n')`

Pour ce qui sort de ces choses basiques : RTFM !

Un exemple

On suppose donné un prédicat `premier` (ou on l'écrit !).

Programme 5 — Écrire les 1000 premiers premiers dans un fichier.

```
fileout = open('premierspremiers.txt', 'w')

nb, k = 0, 2
while nb < 1000:
    if premier(k):
        fileout.write(str(k) + '\n')
        nb += 1
    k += 1

fileout.close()
```

Programme 6 — Et pour les lire en les sommant...

```
filein = open('premierspremiers.txt', 'r')
s = 0
for l in filein.readlines():
    s += int(l)
filein.close()
```

Split, votre nouvel ami

Quoi	Comment
Séparer selon les symboles ':' ou selon les tabulations	`toto.split(':')` `toto.split('\t')`
Virer d'abord les saletés de fin de ligne	`toto.rstrip().split(':')`

Imaginons les 100 premiers nombres premiers écrits dans un fichier, mais regroupés par 10 (c'est l'objet de l'exercice 1), et séparés par des tabulations. Pour les lire et les mettre dans une liste, ça pourrait donner :

- En français : «j'ouvre tel fichier ; pour chaque ligne de ce fichier, je vire les saletés de fin de ligne, je sépare via les tabulations, j'applique la conversion en entier à chaque chaîne de la liste ainsi obtenue... et je concatène le résultat à droite d'une liste de travail.»
- En Python :

Programme 7 — Par blocs de 10

```
filein = open('FilesOut/premiers-par-10.txt', 'r')
lp = []
for l in filein.readlines():
    lp = lp + list(map(int, l.rstrip().split('\t')))
filein.close()
```

La traduction est assez naturelle, non ?

4.2 Exercices

Exercice 15 (Écriture des nombres premiers par 10) *Lire les 1000 premiers nombres premiers dans le fichier* `premierspremiers.txt`*... et les mettre dans une liste. Écrire ensuite dans un nouveau fichier ces 1000 entiers regroupés par 10 et séparés par des tabulations. Ouvrir le fichier créé avec votre éditeur préféré.*

Ce deuxième exercice n'est pas à traiter en priorité !

Exercice 16 *Calculer tous les nombres premiers inférieurs à* 10^6*. Ces nombres seront mis par lignes de dix (séparés par des virgules) dans dix fichiers traitant des tranches de longueur* 10^5*.*

Par exemple, la première ligne de mon fichier `Tranche8.txt` *est*

800011,800029,800053,800057,800077,800083,800089,800113,800117,800119

et la dernière est[5] *:*

899939,899971,899981,

Combien d'entiers inférieurs à 10^6 *sont premiers ?*

Exercice 17 (Lire des grilles de sudoku) *Dans le fichier* `sudoku.txt`*, on trouve 50 grilles de sudoku. Ouvrez ce fichier avec un éditeur, histoire de voir sa tête. Importez les 50 grilles pour en faire une liste de 50 listes de 9 listes de 9 entiers !*

L'exercice suivant est très optionnel...

Exercice 18 (Afficher des grilles) *Affichez les grilles en mode texte. Exemple :*

5. Et oui, je sais que ce serait mieux sans la virgule finale !

```
>>> print(prettyprinting(grids[10]))
 -----------------
|     |1 2 5|4    |
|    8|4    |     |
|4 2  |8    |     |
 -----------------
|  3  |     |  9 5|
|  6  |9   2|  1  |
|5 1  |     |  6  |
 -----------------
|     |    3|  4 9|
|     |    7|2    |
|    1|2 9 8|     |
 -----------------
```

On aura noté que ma fonction `prettyprinting` renvoie *une chaîne, plutôt que de réaliser elle-même l'affichage via* `print` *sans rien renvoyer.*

Exercice 19 *Récupérer les données présentées par exemple sur la page*
`http://agreg.org/ResultatsAlpha2013.html`
et en extraire la liste des candidats admis classés par académie ainsi que le nombre de reçus par académie.
Le tout sans sortir de Python bien entendu...

5 Manipuler des fichiers bitmap

Quoi	**Comment**
THE librairie...	`from PIL import Image`
Ouverture (en lecture)	`Mona = Image.open('FilesIn/joconde.bmp')`
Récupérer la taille	`Mona.size`
Conversion en niveaux (Levels) de gris	`MonaBlack = Mona.convert('L')`
Sauvons Mona	`MonaBlack.save('FilesOut/jocondeBW.bmp')`
Si on préfère le jpg... ou le pdf...	`Mona.save('FilesOut/joconde.jpg')` `Mona.save('FilesOut/joconde.pdf')`

Pour travailler ensuite avec `numpy` (et c'est bien pratique...), on récupère les valeurs des pixels dans un `array`.

Quoi	**Comment**
Mettre les données dans un tableau `numpy`	`bar = numpy.array(MonaBlack)`
Créer une image à partir du tableau de ses pixels puis la sauvegarder	`barfile = Image.fromarray(bar, mode='L')` `barfile.save('....bmp')`

5.1 Exercices sur des images

Faisons comme s'il n'existait pas de méthode `histogram()` faisant ce qu'on imagine...

Exercice 20 (Distribution de niveaux de gris) *L'image noir et blanc de Mona Lisa nous fournit* 350×540 *pixels, qui sont des entiers entre 0 et 255. Déterminer la loi de répartition de ces pixels (bref : une liste de 256 éléments décrivant le nombre de pixels ayant telle valeur). Sauver ces valeurs (ainsi que les sommes cumulées) dans un fichier* `csv`*. Comparer avec la méthode* `histogram()`*. Enfin, regarder l'effet de cette méthode sur une image en couleur.*

Les deux exercices suivants sont réservés à ceux qui connaissent déjà `matplotlib`. Les autres iront voir le corrigé !

Exercice 21 *Représenter l'histogramme de la Joconde avec* `matplotlib`.

Exercice 22 *Comparer l'histogramme de Mona Lisa lorsqu'elle passe du format* `bmp` *au format* `jpg`.

5.2 Floutage

Pour flouter une image, un procédé raisonnable consiste à remplacer chaque pixel par la moyenne des pixels sur un certain voisinage (typiquement, les 9, 25 ou 49 voisins...). Si on travaille en couleurs, on fait la même chose sur chaque composante RGB.

On va travailler avec `numpy`, qui offre ici trois avantages :

- on récupère un tableau au bon format (ses dimensions sont données par le champ `shape` du tableau) ;
- on peut extraire facilement un sous-tableau via `t[i1:i2, j1:j2]`, et on fait facilement la somme d'un tel sous-tableau via `numpy.sum`.
- les calculs sont censés être rapides (mouais...).

Exercice 23 *Écrire une fonction réalisant le floutage d'une image (noir et blanc, ou RGB, composante par composante).*

5.3 Découpage et collage

Exercice 24 *Mettre un bandeau bleu sur les yeux de Mona Lisa.*

Exercice 25 *Échanger deux zones rectangulaires de la Joconde.*

Remerciements

Un grand merci à Judicaël Courant et Stéphane Gonnord pour avoir accepté de partager leur matériel.