Mes premiers pas en python... et les suivants... et ceux d'après.

Mardi 22 et mercredi 23 avril 2014
stephane.gonnord@prepas.org
blog.mp933.fr

1 On se lance

Exercice 1 *TOUS les TP python commencent par cet exercice !*
On commence par lancer Spyder, et TOUT DE SUITE on sauvegarde le fichier de travail sous un nom pertinent (par exemple `'td-debut.py'`, mais certainement pas `'mon 1er td à Luminy !.py'`) à un endroit pertinent (par exemple `Documents/python/stage-luminy-2014/td-initiation/`, mais certainement pas `Documents/` ou `tmp/`). On écrit ensuite `print(42**42)` dans l'éditeur. On sauvegarde et on exécute : un petit passage par F6, puis F5.

Exercice 2 *Tiré de* `projecteuler.net`
Calculer la somme des entiers positifs inférieurs à 1000 qui ne sont divisibles ni par 3 ni par 5.

Allez, cadeau :

$s \leftarrow 0$
pour i *de* 1 *à* 1000 **faire**
  si $i \mod 3 \neq 0$ *et* $i \mod 5 \neq 0$ **alors**
    $s \leftarrow s + i$

ET L'ALGORITHME NE PARLE PAS D'AFFICHAGE. Si on veut la valeur de s (et c'est probablement le cas), on le demande dans l'interpréteur. Ou bien on peut ajouter un `print(s)` dans le fichier `.py`, mais cette chose est bien différente de l'écriture de l'algorithme.

2 Des petites fonctions

2.1 Les yeux fermés

Exercice 3 Écrire un programme testant si une année est bissextile.

Dans le script .py, vous allez donc définir une fonction :

```
def bissextile(n):
    ....
```

Dans la fenêtre d'exécution, testez :

```
>>> bissextile(2014), bissextile(2015), bissextile(2016)
(False, False, True)
>>> bissextile(2000), bissextile(2100)
(True, False)
```

Exercice 4 *Exponentiation*
Pour calculer x^n, on peut peut écrire un programme récursif (idée : $x^{10} = x \times x^9$; subtil, non ?) ou itératif :

Entrées : x, n
$res \leftarrow 1$
pour i *de* 1 *à* n **faire**
  $res \leftarrow res \times x$
Résultat : res

1. Programmer (au moins) l'un de ces deux points de vue : `def expo(x,n): ...`
2. Combien vaut 3^{19} ?

3. Combien de mutiplications sont effectuées pour calculer x^n ?
4. En notant que $x^{42} = (x^{21})^2$ et $x^{19} = x \times (x^9)^2$, on obtient un algorithme sensiblement meilleur... programmez-le, vérifiez que 3^{19} vaut toujours la même chose, et évaluez le nombre de multiplications nécessaires pour calculer x^{1023}.

Exercice 5 *Fonction factorielle*
Pour calculer $n!$, on a à nouveau le choix entre le point de vue récursif ($10! = 10 \times 9!$) ou itératif :

Entrées : n
$res \leftarrow 1$
pour i *de* 1 *à* n **faire**
  $res \leftarrow res \times i$
Résultat : res

1. Programmer une ou deux fonctions « factorielle ».
2. Vérifier que 10! vaut la bonne valeur... qu'on peut par exemple demander à la fonction `factorial` de la bibliothèque `math`.

2.2 Plus élaboré

Exercice 6 *Fibonacci*
La suite de Fibonacci est définie par $f_0 = 0$, $f_1 = 1$, et $f_{n+2} = f_n + f_{n+1}$ pour tout $n \in \mathbb{N}$. On peut choisir entre le point de vue récursif et le point de vue itératif pour programmer son calcul :

Entrées : n
si $n \leqslant 1$ **alors**
  Résultat : n
$(a, b) \leftarrow (0, 1)$
pour k *de* 2 *à* n **faire**
  $(a, b) \leftarrow (b, a + b)$ // *On calcule les deux termes courants :* f_{k-1} *et* f_k.
Résultat : b

1. Écrire un programme prenant en entrée un entier n et renvoyant f_n.
2. A-t-on bien $f_{10} = 55$?
3. Que vaut f_{100} ?

Exercice 7 *Somme des décimales d'un entier*
Pour $n \in \mathbb{N}$, on note $\varphi(n)$ la somme de ses chiffres dans la représentation décimale. Par exemple, $\varphi(42) = 6$ et $\varphi(2013) = 6$. Comme souvent, on a une attaque récursive : pour trouver $\varphi(2013)$, il suffit d'ajouter 3 à $\varphi(201)$. Mais on a également une attaque itérative consistant à avoir d'un côté la somme s déjà calculée, et de l'autre l'entier r qui reste à traiter :

Entrées : n
$(s, r) \leftarrow (0, n)$
tant que $r > 0$ **faire**
  $s \leftarrow s + r \mod 10$ // *On ajoute à* s *la dernière décimale de* r.
  $r \leftarrow r/10$ // *On efface la dernière décimale de* r.
Résultat : s

1. Écrire une (ou deux) fonctions calculant φ.
2. Vérifier que $\varphi(100!) = 648$.
3. Que vaut $\varphi(500!)$?

Exercice 8 *Primalité*
Une façon élémentaire de tester le caractère premier ou pas d'un entier n consiste à voir s'il est divisible ou non par en entier compris (au sens large) entre 2 et $\sqrt{n}$.

Entrées : n
$t \leftarrow 2$ // *La valeur par laquelle on va tester la divisibilité*
tant que $t \leqslant \sqrt{n}$ **faire**
  si $n \mod t = 0$ **alors**
    Résultat : Faux
  $t \leftarrow t + 1$
Résultat : Vrai

1. Écrire un programme testant la primalité des entiers.
2. Combien d'entiers entre 2 et 1000 sont premiers ?
3. Sans lancer votre programme, que pensez-vous qu'il va se passer si on change 10^3 en 10^6 dans la question précédente ?
4. Et si on change 10^6 en 10^7 ? [1]

2.3 Du rabe

Exercice 9 *Algorithme d'Euclide*
Pour calculer le pgcd de deux entiers, on ne calcule jamais leurs factorisations ! Le point crucial est que si $a > b > 0$, alors $pgcd(a, b) = pgcd(b, a \mod b)$, et ce procédé permet de tomber rapidement sur un pgcd de la forme $pgcd(a, 0) = a$. On peut appliquer ce principe récursivement ou itérativement (c'est ce point de vue qu'on appelle en général l'algorithme d'Euclide).

Entrées : a, b
$(g, d) \leftarrow (\max(a, b), \min(a, b))$ // *Quand je l'exécute à la main, le plus gros est à gauche*
tant que $d > 0$ **faire**
  $(g, d) \leftarrow (d, g \mod d)$
Résultat : g

1. Écrire un programme (ou deux !) calculant le pgcd de deux entiers positifs.
2. Plus difficile : écrire un programme retournant les coefficients de Bezout, c'est-à-dire deux entiers relatifs u et v tels que $au + bv = pgcd(a, b)$.

Exercice 10 *La suite de Syracuse*
Si n est un entier strictement positif, la suite de premier terme $u_0 = n$ et vérifiant la relation de récurrence... finit [2] par tomber sur 1... pour ne plus le quitter (modulo le cycle $1 \to 4 \to 2 \to 1 \to \cdots$). On note alors s_n le premier entier tel que $u_{s_n} = 1$. Par exemple, $s_1 = 0$, $s_1 = 1$ et $s_{10} = 6$.

1. Écrire une fonction calculant s_n.
2. Donner les valeurs de s_{15} et s_{127}.
3. Quelle est le maximum des s_n pour $n \leqslant 10^3$? Et $n \leqslant 10^6$?

Exercice 11 *Quel jour sommes-nous ?*
Rappel : 2016 sera bissextile, mais pas 2100 (multiple de 100), alors que 2400 le sera (multiple de 400).

1. Sachant que le premier janvier 2014 est tombé un mercredi, écrire une fonction retournant le jour dans la semaine où tombe le premier janvier d'une année. On prendra comme convention : 1 pour lundi, 2 pour mardi... jusqu'à 7 pour dimanche. Vérifier avec les exemples suivants :

```
stephane@euler:~$ cal janvier 2042
    January 2042
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
...
stephane@euler:~$ cal janvier 1917
    January 1917
```

1. Le TD sur les tableaux reprendra cette question. Quelle coordination dans ce stage !
2. Enfin, on pense que c'est vrai !

```
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
...
```

2. Écrire une fonction prenant en entrée une date (triplet d'entiers jour/mois/année) et retournant le jour dans la semaine correspondant à cette date.

```
>>> jour(9,11,1971)
2
```

3. Quel jour sommes-nous ?

3 Manipulation de listes/tableaux

Exercice 12 Lancer Spyder, et sauvegarder le fichier de travail sous un nom pertinent (par exemple `'td-tableaux.py'`, mais certainement pas `'le super td sur les tableaux à Luminy.py'`) à un endroit pertinent (par exemple `'Documents/python/stage-luminy-2014/tableaux/'`, mais certainement pas `'Documents/'` ou `'tmp/'`).

On écrit ensuite `print(42**42)` dans l'éditeur. On sauvegarde et on exécute : F6 puis F5.

3.1 De la théorie à deux balles

- Les tableaux, ou listes, sont les « vecteurs » de python : on stocke « en ligne » les données ; on a accès à la k-ème en temps constant [3]
- On définit/lit/modifie un tableau de la façon suivante :

```
>>> t = [12, 7, -15, 'a']
>>> len(t)
4
>>> t[1]
7
>>> t[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> t[-1]
'a'
>>> t[2] = 7
>>> t
[12, 7, 7, 'a']
```

- Le *slicing* [4] permet de récupérer des morceaux de tableaux. Attention au bord droit, qui est exclu.

```
>>> t[0:2]
[12, 7]
>>> t[1:3]
[7, 7]
>>> t[:2]
[12, 7]
>>> t[1:]
[7, 7, 'a']
>>> t[:]
[12, 7, 7, 'a']
```

  À chaque fois, la liste retournée est une nouvelle liste. La dernière commande permet donc de faire une copie de liste [5].

3. Au moins en première approximation.
4. Ce qui doit donner quelque chose comme *découpage en tranche* en français.
5. L'orateur vous parlera certainement de `deepcopy` en faisant des dessins abscons avec des tas de flèches ; plissez les yeux, prenez un air intelligent, puis hochez la tête d'un air entendu !

– Les *listes par compréhension* sont des moyens diaboliques pour créer facilement des listes. Un peu à la `seq` en maple.
```
>>> [i**2 for i in range(-2,5)]
[4, 1, 0, 1, 4, 9, 16]
>>> [i**2 for i in range(-2,5) if abs(i)>1]
[4, 4, 9, 16]
```
– Une autre façon efficace de créer des listes : `[0] * 10`. Efficace mais dangereux (voir plus tard). Je vous conseille plutôt : `[0 for i in range(10)]`, ou `[0 for _ in range(10)]` si vous ne voulez pas donner de nom à une variable muette !
– Quand on donne un tableau en entrée à un programme, est-ce par valeur ou par adresse ? En fait, formellement, la valeur du tableau est son adresse ! Le comportement est donc le « par adresse » au sens usuel : le programme va (pouvoir) modifier le tableau qu'on lui a donné.
– En fait, les tableaux sont redimensionnables (on peut ajouter un élément pour un prix modique [6]). C'est ce qui fait qu'ils peuvent être utilisés comme des listes/piles, via des méthodes `pop` et `append` [7].

L'exercice suivant est fondamental...

Exercice 13 *On peut toujours rêver*
Promettre de ne jamais nommer une liste `l`.

3.2 Des parcours de base

Voici deux façons de parcourir un tableau : à l'ancienne (via les positions), ou bien à la python (via une itération directe sur le tableau) :

```
for i in range(len(t)):
    gnagna sur t[i]
...
for x in t:
    gnagna sur x
```

La deuxième façon peut vous sembler étrange, mais après quelques heures de python, vous aurez oublié qu'à une époque ancienne, vous utilisiez la première !

On commence par définir deux listes « jouet » sur lesquelles on testera les différentes fonctions de façon systématique.

```
jouet1 = [1, 4, 12, -5, -3]
jouet2 = [i**2 for i in range(50)]
```

Exercice 14 Écrire une fonction calculant la somme des éléments d'un tableau.

Entrées : t
$s \leftarrow 0$
pour x *dans* t **faire**
  $s \leftarrow s + x$
Résultat : s

Exercice 15 Écrire une fonction calculant le maximum des éléments d'un tableau supposé non vide.
Entrées : t
$m \leftarrow t[0]$ // *Le maximum provisoire*
pour x *dans* t **faire**
  si $x > m$ **alors**
    $m \leftarrow x$
Résultat : m

6. Constant en complexité amortie.
7. Note de Guido van Rossum en 1997 : *To implement a stack, one would need to add a list.pop() primitive (and no, I'm not against this particular one on the basis of any principle). list.push() could be added for symmetry with list.pop() but I'm not a big fan of multiple names for the same operation – sooner or later you're going to read code that uses the other one, so you need to learn both, which is more cognitive load.*

Même chose ensuite avec la *position* du maximum. En cas d'égalité, comment faire pour choisir la première position des ex-æquo ? Et la dernière ?

EXERCICE 16 *C'est un des items du programme*
Écrire un programme calculant la moyenne et la variance d'un tableau [8]

EXERCICE 17 Écrire un programme testant si tous les éléments d'un tableau sont positifs (au sens large).

Entrées : t
pour x *dans* t **faire**
  si $x < 0$ **alors**
    Résultat : False // *Il faut sortir*
Résultat : True

EXERCICE 18 Écrire un programme testant si un tableau est croissant (au sens large).

EXERCICE 19 Reprendre les deux exercices suivants et, si ce n'était pas déjà le cas, éliminer les `return` multiples : il ne doit y avoir que une fois le mot-clé `return`.
Attention, il s'agit (bonus) de sortir dès que possible, tout de même !

À partir de maintenant, vous pouvez vous autoriser... ou contraire vous interdire ces `return` multiples : il y a différentes religions sur le sujet [9].

EXERCICE 20 Écrire un programme calculant le nombre d'inversions d'un tableau, c'est-à-dire le nombre de couples (i, j) tels que $0 \leqslant i < j < n$ tels que $t[i] > t[j]$, avec n la longueur de t.

3.3 Recherches diverses

EXERCICE 21 Écrire un programme prenant en entrée x et un tableau t, et renvoyant `True` ou `False` selon que x appartient ou non à t.

EXERCICE 22 Écrire un programme prenant en entrée x et un tableau t, et renvoyant la première position de x dans t si x est présent, et `None` sinon.
Comment faire pour renvoyer la dernière position ? Donner deux possibilités !

EXERCICE 23 Écrire un programme prenant en entrée x et un tableau t, et renvoyant la liste (éventuellement vide) des positions de x dans t.
Pour ajouter un élément à une liste, exécuter `t.append(x)` *(et surtout pas* `t = t.append(x)`*)*

EXERCICE 24 *Recherche dichotomique ; explicitement au programme.*
Écrire un programme prenant en entrée x et un tableau t *supposé trié dans l'ordre croissant*, et testant l'appartenance de x à t par une recherche dichotomique.
Quel est la complexité [10] de cette recherche ?

EXERCICE 25 *Recherche d'un sous-mot ; explicitement au programme.*
Écrire un programme prenant en entrée un motif m et un mot w, et testant si le motif m est présent dans w. *On autorise ici le slicing, c'est-à-dire des comparaisons du type* `w[i,j]=m`.

```
>>> sous_mot1('titi','tructititoto')
4
>>> sous_mot1('tagada','tructititoto')
>>>
```

EXERCICE 26 *Recherche d'un sous-mot, sans slicing*
Reprendre l'exercice précédent, mais en s'interdisant le slicing.

8. Enfin.. son estimateur biaisé ; bref : $\frac{1}{n}$ plutôt que $\frac{1}{n-1}$!
9. Sur ce point (également) je suis athée !
10. Nombre de comparaisons effectuées.

3.4 Tableaux bidimensionnels

Les éléments d'un tableau peuvent être des tableaux ! Cela permet de construire par exemple des matrices.

```
>>> m=[[1, 2, 3], [4, 5, 6]]
>>> len(m)
2
>>> len(m[0])
3
>>> m[1,2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: list indices must be integers, not tuple
>>> m[1][2]
6
```

En agitant les bras et avec des tas de dessins, l'orateur va vous expliquer ce qui se passe ici :

```
>>> m=[[0]*3]*2
>>> m
[[0, 0, 0], [0, 0, 0]]
>>> m[0][1] = 2
>>> m
[[0, 2, 0], [0, 2, 0]]
```

et ce qui se passe là :

```
>>> m = [[0] * 3 for i in range(2)]
>>> m
[[0, 0, 0], [0, 0, 0]]
>>> m[0][1] = 2
>>> m
[[0, 2, 0], [0, 0, 0]]
```

C'est ce type de problêmes qui font que la recopie d'un tableau est une opération hautement non triviale !

Si l'agitation de bras n'a as été suffisante vous pouvez aussi allez voir le très pédagogique

`http://www.pythontutor.com/visualize.html`

Exercice 27 Écrire des programmes :
- donnant les dimensions d'une matrice ;
- construisant une matrice nulle à dimensions imposées ;
- construisant la matrice identité I_n, avec n donné.

Exercice 28 Écrire un programme prenant en entrée des dimensions (n, p), une fonction f, et renvoyant la matrice (n, p) de terme général $f(i, j)$. On fera attention à mettre en position $(0, 0)$ ce que le matheux appelle $m_{1,1}$, donc $f(1, 1)$.
Je sais, c'est pénible, mais c'est ainsi.

Reprendre l'exercice précédent à l'aide de cette nouvelle fonction générique.

Exercice 29 Écrire un programme testant si une matrice est symétrique.

Exercice 30 Écrire des programmes calculant la transposée d'une matrice, puis la somme et le produit de deux matrices.

Exercice 31 *Explicitement au programme ; mériterait du temps et quelques étapes intermédiaires !* Écrire un programme inversant une matrice par pivot de Gauss.

On trouve sur de très bons sites web des TPs entièrement consacrés à ce point...

3.5 Un peu de rabe

Exercice 32 *Le crible d'Ératosthène*
Pour déterminer l'ensemble des nombre premiers majorés par n, on peut créer un tableau de longueur $n+1$ (pour ne pas être embêté avec les décalages d'indices) constitué de la valeur `True` (tant qu'on a pas prouvé que n est composé, il est supposé premier), puis en « criblant » :

- tous les multiples (stricts) de 2 sont composés : on les passe à `False` dans le tableau ;
- tous les multiples (stricts) de 3 sont composés : on les passe à `False` dans le tableau ;
- pour 4, qui est composé (puisque `t[4] == False`), on laisse tomber (les multiples ont déjà été rayés) ;
- tous les multiples (stricts) de 5 sont composés : ...
- pour 6, qui est composé...
- etc.

On continue tant qu'« on » (ce par quoi on crible) n'a pas dépassé $\sqrt{n}$.

1. Écrire un programme prenant en entrée n et renvoyant la liste des entiers majorés par n qui sont premiers.
2. Quelle est la complexité de ce programme ?
3. Combien d'entiers majorés par 10^7 sont premiers ?

Exercice 33 *Permutations dans des tableaux*
On peut voir une permutation de $[\![0, n-1]\!]$ comme une liste de n entiers... Écrire alors :

- `verifier_permutation` qui vérifie si une liste correspond bien à une permutation.
- `inverse` qui calcule la bijection réciproque d'une permutation.
- `compose` qui calcule la composée de deux permutations.

On pourra aussi jouer avec les décompositions en cycle, l'ordre, la signature, etc.

Exercice 34 *Tri par sélection*
Rappel du principe : on sélectionne le plus gros élément, et on le place à la fin (via un échange). Ensuite, on sélectionne le plus gros parmi les $n-1$ premiers, qu'on place en position $n-1$ via un échange ; etc...

Programmer le tri par sélection, dans une fonction réalisant le tri en place. Cette fonction ne doit rien renvoyer, mais modifier la liste donnée en paramêtre [11].

```
>>> t = [12, 5, -3, 42, 1515, 17, 3]
>>> tri_selection(t)
>>> t
[-3, 3, 5, 12, 17, 42, 1515]
```

11. On parle d'*effet de bord*