

Arithmétique et listes en compréhension

Cette fiche a été rédigée par Alain Busser. Il enseigne au lycée Roland-Garros du Tampon. Il est aussi animateur à l'IREM de La Réunion et créateur du langage de programmation Sophus.

Introduction : la fonction modulo

La fonction modulo (notée %, disponible dans la **Toolbox**) donne le reste dans la division euclidienne du nombre situé avant le %, par le second nombre. Par exemple, si on effectue la division de 23 par 10, le reste est 3, ce qui se note $23\%10=3$. Cela suppose que les opérandes sont des entiers. La divisibilité peut se tester à l'aide de cette opération :

On dit que a est divisible par b , ou que a est un multiple de b , si $a\%b==0$.

Par exemple, pour obtenir la liste des multiples de 3 entre 0 et 20, on peut écrire la liste en extension : `[0,3,6,9,12,15,18]` ou la décrire en compréhension : `[t for t in range(20) if t%3==0]`. On dit qu'on a filtré la liste des entiers inférieurs à 20, par la fonction booléenne "est multiple de 3".

Cette notion est fondamentale en programmation fonctionnelle et dans les algorithmes de Big Data.

Exercices préliminaires

- Décrire en compréhension la liste des entiers pairs de 2 chiffres.
- Décrire en compréhension la liste des nombres inférieurs à 100 qui sont multiples, ou bien de 5, ou bien de 7.
- Décrire en compréhension la liste **z f i z z b u z** des entiers inférieurs à 100 qui sont multiples de 5 ou de 7 mais pas de 5 et 7 en même temps.
- Si le temps le permet, programmer le crible d'Eratosthène avec des compréhensions.

Exemples

Les compréhensions permettent de programmer facilement les fonctions suivantes :

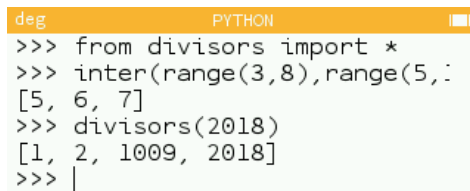
- **inter**, qui accepte deux listes $l1$ et $l2$ en entrée, et renvoie l'intersection des deux listes, c'est-à-dire la liste des éléments communs à $l1$ et $l2$;
- **divisors**, qui accepte un entier n en entrée, et renvoie la liste de ses diviseurs.

Pour les tester, créer un module Python appelé `divisors.py` et y entrer ceci :

```
1 def inter(l1,l2):  
2     return [x for x in l1 if x in l2]  
3 def divisors(n):  
4     return [d for d in range(1,n+1) if n%d==0]
```

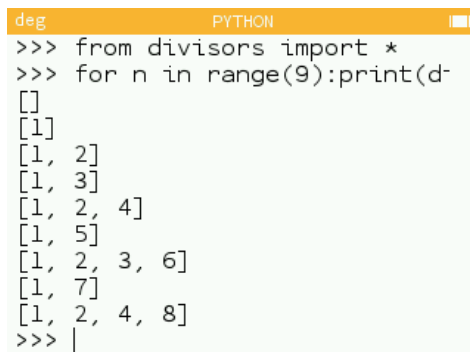
Ensuite, retourner dans la console interactive et y entrer `inter(range(3,8),range(5,13))` pour savoir quelle est l'intersection des listes `[3,4,5,6,7]` et `[5,6,7,8,9,10,11,12]`.

Pour savoir quels sont les diviseurs de 2018, faire `divisors(2018)`.



```
>>> from divisors import *  
>>> inter(range(3,8),range(5,13))  
[5, 6, 7]  
>>> divisors(2018)  
[1, 2, 1009, 2018]  
>>> |
```

On peut voir à quoi ressemblent des listes de diviseurs pour plusieurs entiers, en appelant la fonction `print()` dans une boucle comme `for n in range(9):print(divisors(n))` laquelle produit l'effet ci-dessous.



```
>>> from divisors import *  
>>> for n in range(9):print(d-  
[]  
[1]  
[1, 2]  
[1, 3]  
[1, 2, 4]  
[1, 5]  
[1, 2, 3, 6]  
[1, 7]  
[1, 2, 4, 8]  
>>> |
```

On voit que l'algorithme est pris en défaut pour 0, qui est affiché n'ayant aucun diviseur alors qu'en réalité c'est tout le contraire : 0 est divisible par tous les autres entiers. Ceci dit, il ne serait pas prudent de demander à la NumWorks d'afficher une liste de taille infinie ! Ensuite, 1 est clairement le seul entier n'ayant qu'un seul diviseur, les autres sont tous divisibles par au moins 1 et eux-mêmes. Ce qui amène à la notion de primalité.

Exercice

Rédiger un test de primalité sous forme d'une fonction **estPremier(*n*)** acceptant en entrée un entier *n*, et renvoyant **True** si *n* est premier, et **False** sinon. On peut faire appel à la fonction **len(*L*)** qui accepte une liste *L* en entrée et renvoie le nombre de ses éléments, ainsi qu'à l'une des fonctions précédemment définies.

Autres exercices

On aborde dans la série d'exercices suivants, la notion de diviseurs communs à deux entiers.

1. Écrire une fonction Python **cd(*a*,*b*)** qui accepte deux entiers *a* et *b* en entrée et qui renvoie la liste des diviseurs communs à *a* et *b*. On peut utiliser la fonction **inter()** définie au début de cette activité.
2. En déduire un algorithme "naïf" de calcul du pgcd, qui est le plus grand élément de **cd(*a*,*b*)**, comme son nom "plus grand commun diviseur" l'indique. On pourra utiliser la fonction **max(*L*)** qui accepte en entrée une liste *L* de nombres et renvoie le plus grand élément de cette liste. On demande une fonction **pgcd(*a*,*b*)** qui accepte en entrée deux entiers *a* et *b*, et renvoie leur pgcd.
3. En déduire également un test **premiersEntreEux(*a*,*b*)** qui accepte en entrée deux entiers *a* et *b* et renvoie **True** si *a* et *b* sont premiers entre eux et **False** s'ils ont un diviseur commun autre que 1. Là encore, la fonction **len** peut être utile, puisque le fait que deux entiers soient premiers entre eux est lié au nombre de leurs diviseurs communs.
4. Déduire de la fonction précédente, une fonction **eulerphi(*n*)** où *n* est un entier, renvoyant le nombre d'entiers inférieurs à *n* et premiers avec *n*. Par exemple, les nombres inférieurs à 8 et premiers avec 8 sont 1, 3, 5 et 7 ce qui fait que **eulerphi(8)==4**.
5. On dit qu'un entier est parfait si la somme de ses diviseurs est égale à son double. Par exemple, les diviseurs de 6 sont 1, 2, 3 et 6 et $1+2+3+6=12$ qui est le double de 6, donc 6 est parfait. Écrire un test de perfection sous la forme d'une fonction Python **estParfait(*n*)** acceptant un entier *n* en entrée et renvoyant **True** si *n* est parfait et **False** sinon. On pourra utiliser la fonction **sum(*L*)** acceptant en entrée une liste d'entiers *L* et renvoyant la somme des éléments de cette liste. On a vu que 6 est un nombre parfait. Quels sont les deux suivants ?