	<h1 style="text-align: center;">INFORMATIQUE ET SCIENCES DU NUMERIQUE</h1> <p style="text-align: center;">Lycée François Villon Beaugency</p>	Cours <input checked="" type="checkbox"/>
Spécialité ISN	<h2 style="text-align: center;">Introduction aux fonctions Python</h2>	page 1 / 4

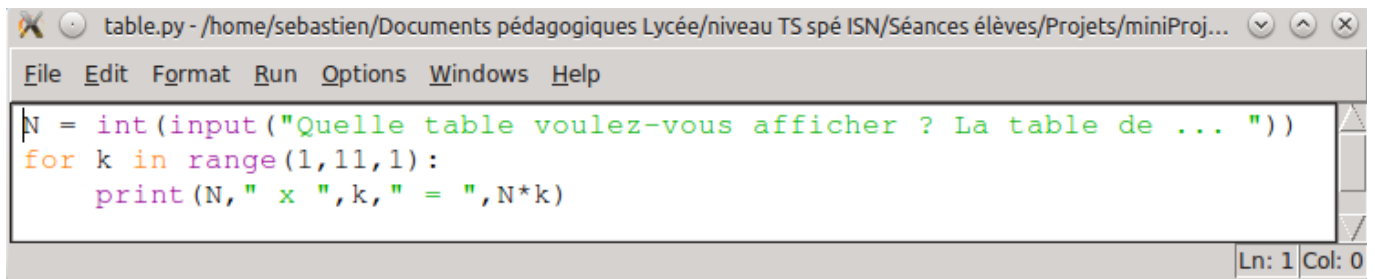
Un programme Python peut parfois être long. Plus il est long, moins il est facile à lire par une personne qui ne l'a pas écrit. Et s'il est très long, il peut arriver que même son auteur s'y perde quand arrive l'inévitable débuggage...

Il est donc nécessaire de prendre de bonnes habitudes de rédaction des programmes.

Pour cela, Python est bien conçu puisqu'il permet d'écrire des **fonctions** : ce sont des « sous-programmes » utilisables à tout moment dans la partie principale du programme.

L'avantage : on les écrit avant, bien séparés du reste : on segmente donc l'écriture du programme qui devient plus détaillé et plus lisible (mais pas forcément plus court...)

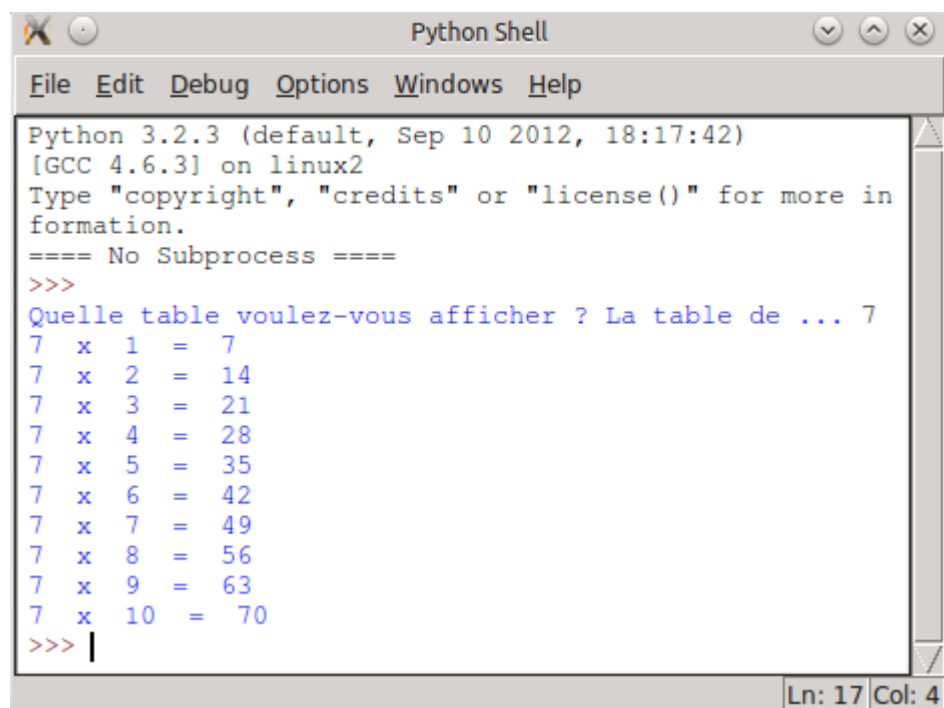
Un premier exemple : on souhaite écrire un programme qui demande à l'utilisateur quelle table de multiplication il veut afficher. On a déjà écrit ce programme, qui donnait :



```

N = int(input("Quelle table voulez-vous afficher ? La table de ... "))
for k in range(1,11,1):
    print(N, " x ", k, " = ", N*k)
  
```

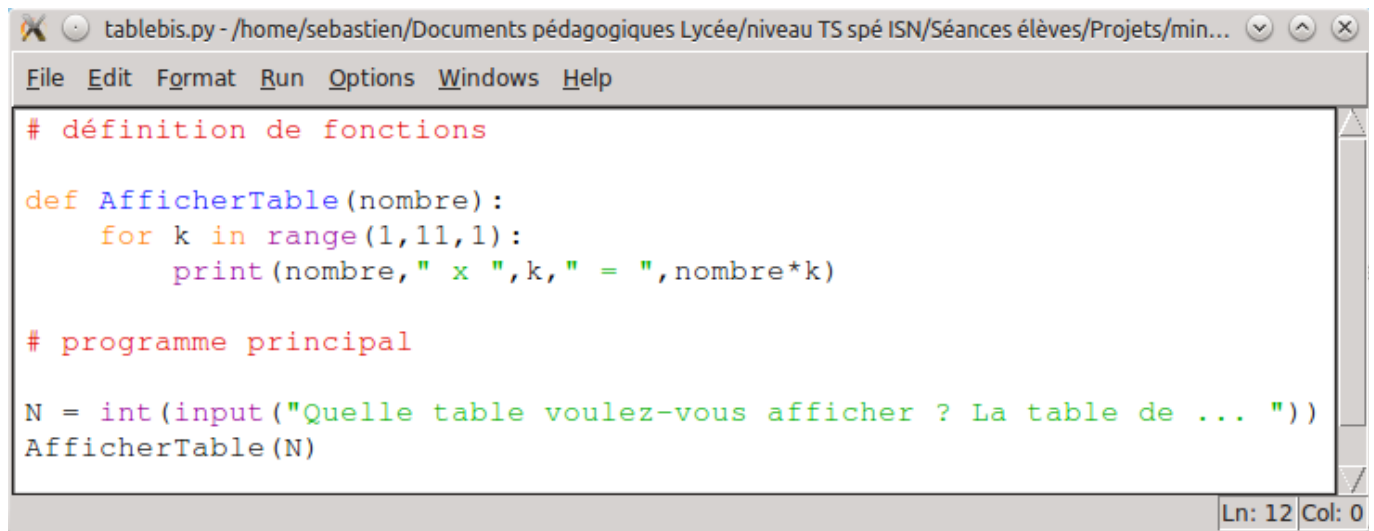
Avec pour résultat :



```

Python 3.2.3 (default, Sep 10 2012, 18:17:42)
[GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more in
formation.
==== No Subprocess ====
>>>
Quelle table voulez-vous afficher ? La table de ... 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
>>>
  
```

Avec Python, on peut envisager ce type de rédaction :



```
# définition de fonctions

def AfficherTable(nombre):
    for k in range(1,11,1):
        print(nombre, " x ", k, " = ", nombre*k)

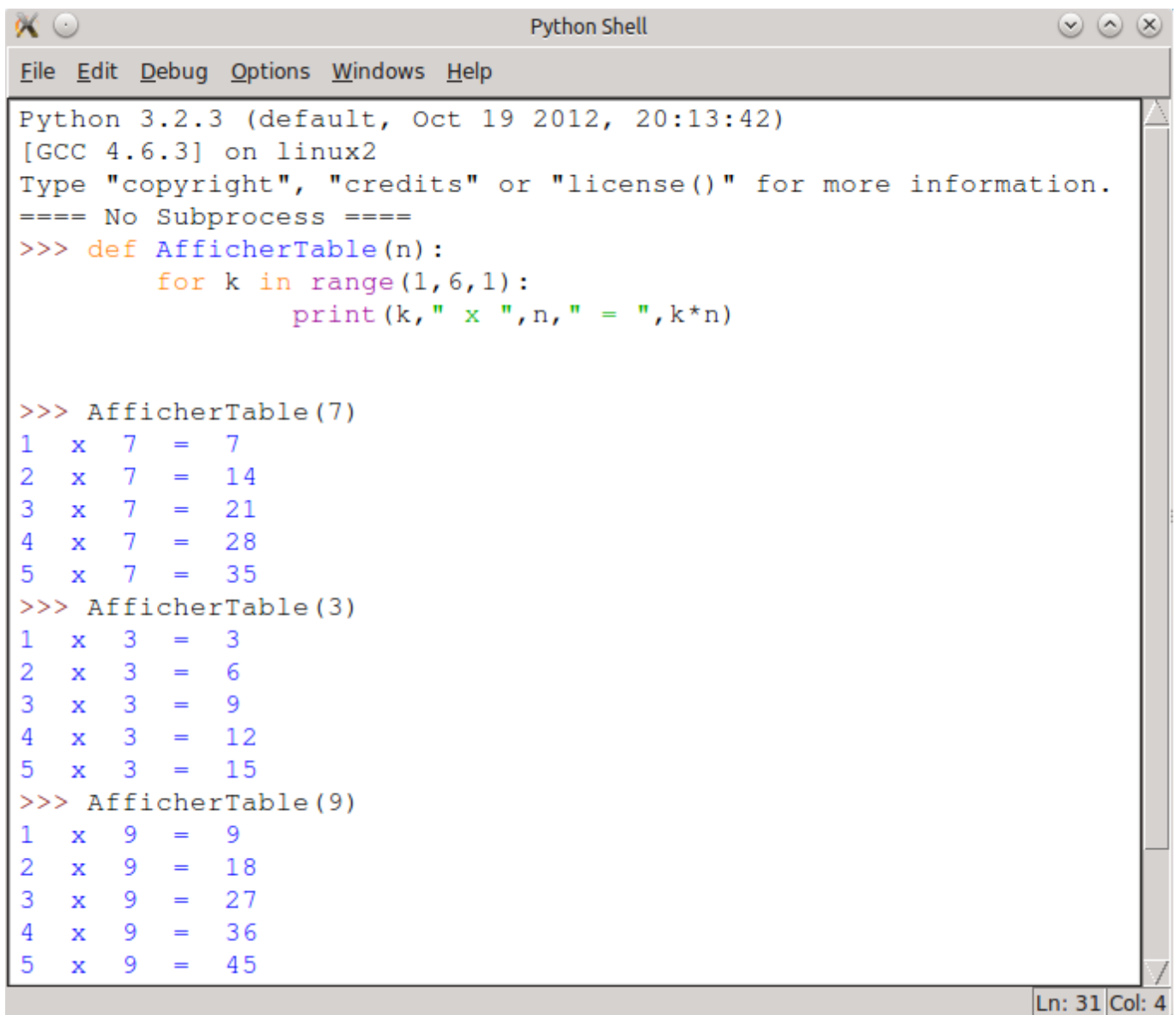
# programme principal

N = int(input("Quelle table voulez-vous afficher ? La table de ... "))
AfficherTable(N)
```

Ln: 12 Col: 0

Le résultat est exactement le même que pour la version 1. La différence est pourtant de taille. Le **programme principal** est bien **plus lisible**. Pas plus court, mais **plus lisible**...

On peut aussi définir cette fonction **AfficherTable(n)** dans la fenêtre d'exécution immédiate, et l'appeler autant qu'on veut :



```
Python 3.2.3 (default, Oct 19 2012, 20:13:42)
[GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more information.
==== No Subprocess ====
>>> def AfficherTable(n):
        for k in range(1,6,1):
            print(k, " x ", n, " = ", k*n)

>>> AfficherTable(7)
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
>>> AfficherTable(3)
1 x 3 = 3
2 x 3 = 6
3 x 3 = 9
4 x 3 = 12
5 x 3 = 15
>>> AfficherTable(9)
1 x 9 = 9
2 x 9 = 18
3 x 9 = 27
4 x 9 = 36
5 x 9 = 45
```

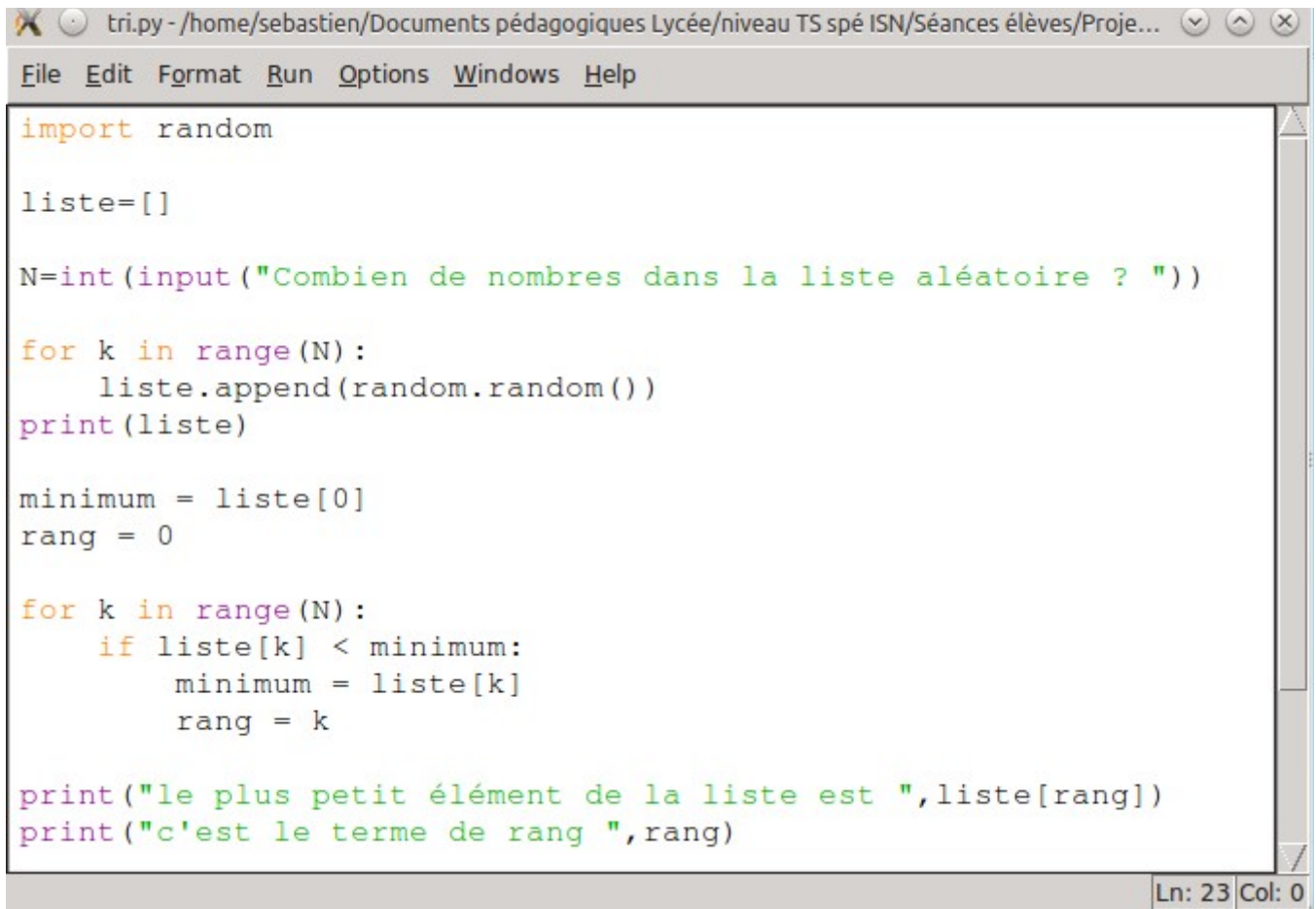
Ln: 31 Col: 4

Un deuxième exemple : une première approche des tris :

Dans cet exemple on cherche à :

- fabriquer une liste aléatoire (un certain nombre de réels entre 0 et 1) ;
- rechercher dans cette liste le plus petit élément, et son rang.

Voici un programme organisé « au fil des idées » :



```
tri.py - /home/sebastien/Documents pédagogiques Lycée/niveau TS spé ISN/Séances élèves/Proje...
File Edit Format Run Options Windows Help

import random

liste=[]

N=int(input("Combien de nombres dans la liste aléatoire ? "))

for k in range(N):
    liste.append(random.random())
print(liste)

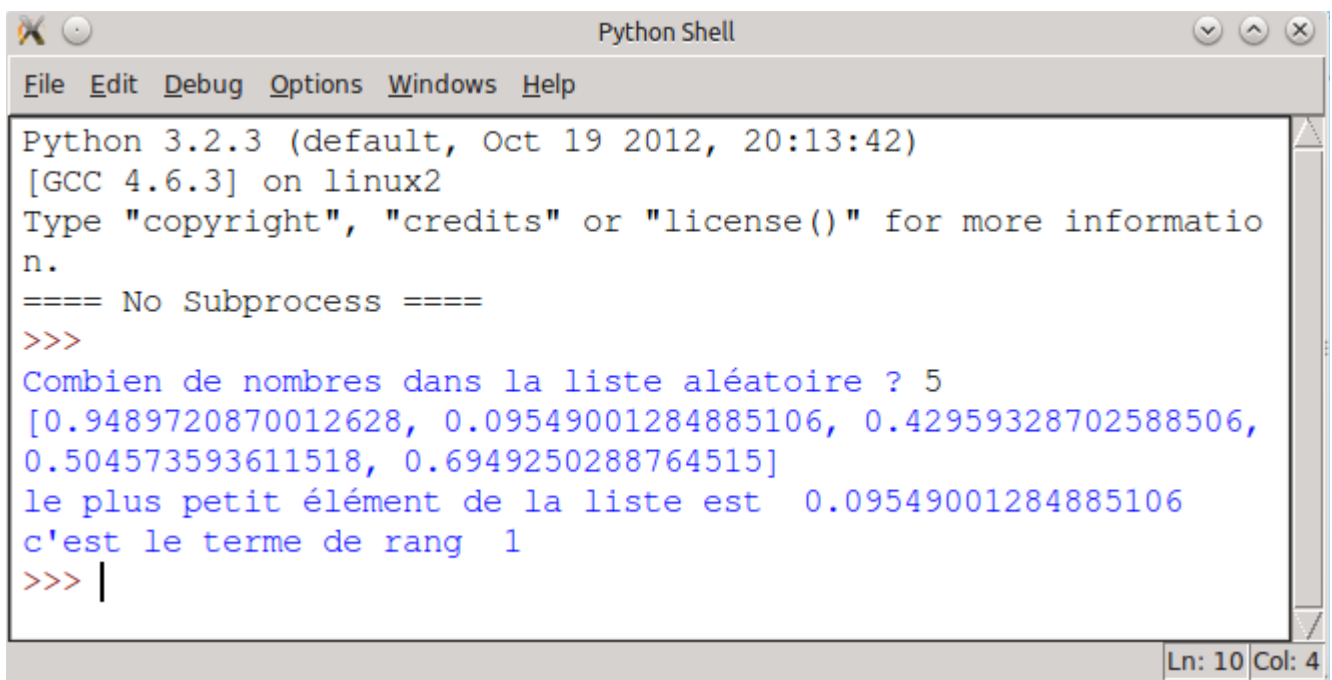
minimum = liste[0]
rang = 0

for k in range(N):
    if liste[k] < minimum:
        minimum = liste[k]
        rang = k

print("le plus petit élément de la liste est ",liste[rang])
print("c'est le terme de rang ",rang)

Ln: 23 Col: 0
```

Son résultat :

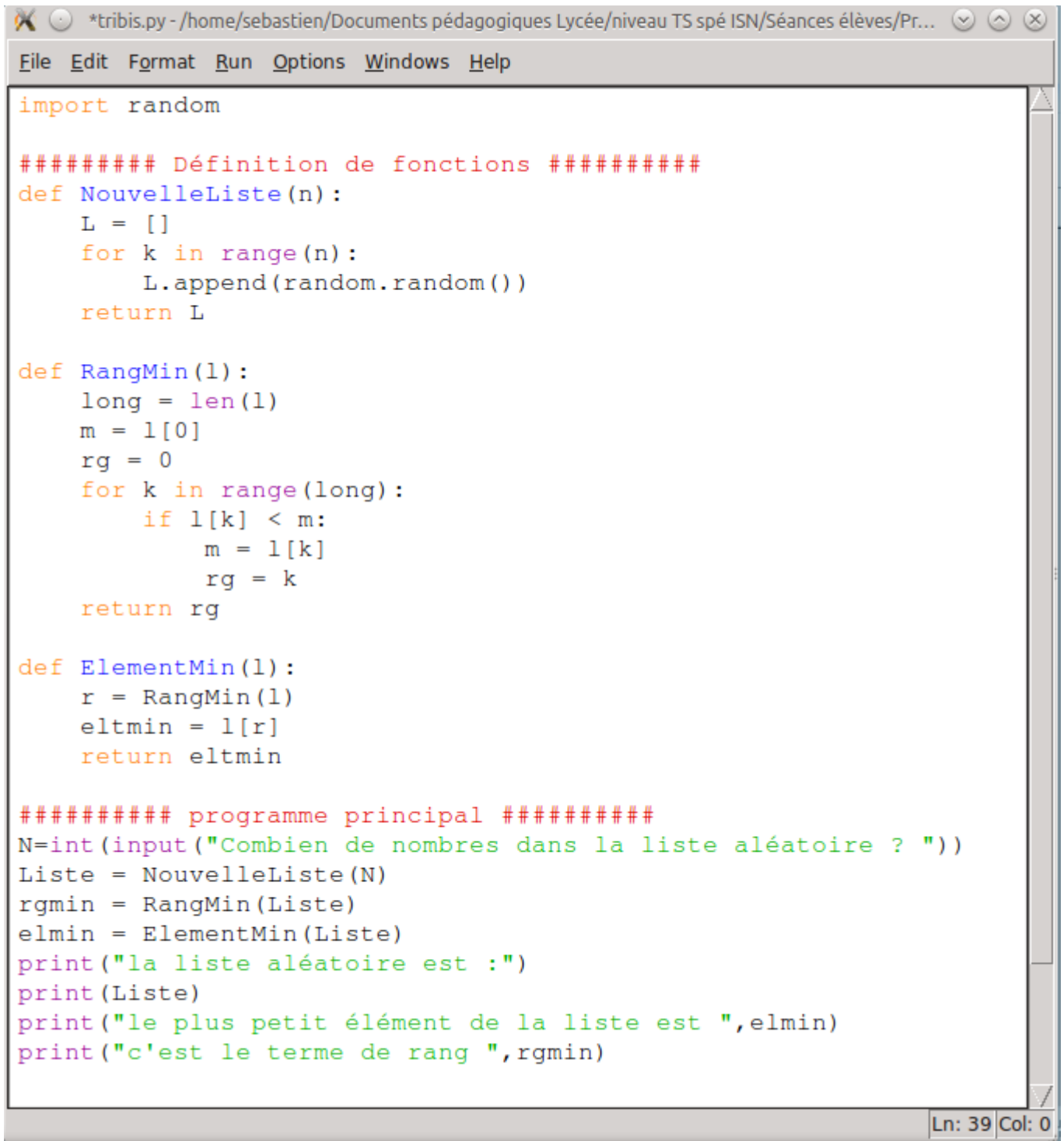


```
Python Shell
File Edit Debug Options Windows Help

Python 3.2.3 (default, Oct 19 2012, 20:13:42)
[GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more informatio
n.
==== No Subprocess ====
>>>
Combien de nombres dans la liste aléatoire ? 5
[0.9489720870012628, 0.09549001284885106, 0.42959328702588506,
0.504573593611518, 0.6949250288764515]
le plus petit élément de la liste est  0.09549001284885106
c'est le terme de rang  1
>>> |

Ln: 10 Col: 4
```

Voici une version « fonctionnelle » du même programme :



```
import random

##### Définition de fonctions #####
def NouvelleListe(n):
    L = []
    for k in range(n):
        L.append(random.random())
    return L

def RangMin(l):
    long = len(l)
    m = l[0]
    rg = 0
    for k in range(long):
        if l[k] < m:
            m = l[k]
            rg = k
    return rg

def ElementMin(l):
    r = RangMin(l)
    eltmin = l[r]
    return eltmin

##### programme principal #####
N=int(input("Combien de nombres dans la liste aléatoire ? "))
Liste = NouvelleListe(N)
rgmin = RangMin(Liste)
elmin = ElementMin(Liste)
print("la liste aléatoire est :")
print(Liste)
print("le plus petit élément de la liste est ",elmin)
print("c'est le terme de rang ",rgmin)
```

Ln: 39 Col: 0

Vous aurez remarqué que dans la définition des fonctions, il y a des variables dites « locales ».

Par exemple, l'instruction `Liste = NouvelleListe(N)` utilise une variable `N`, et cet appel de la fonction `NouvelleListe` va donc se faire avec la variable locale `n` qui prendra la valeur de `N` du programme principal...