

# **PYTHON en MIASHS L1 & L2**

**Dominique Gonzalez**  
Université Lille3-Charles de Gaulle

**PYTHON en MIASHS L1 & L2**  
par Dominique Gonzalez

Publié Mars 2005  
Copyright © 2005 D.Gonzalez

Ce document est soumis à la licence GNU FDL. Permission vous est donnée de distribuer, modifier des copies de ces pages tant que cette note apparaît clairement.

# Table des matières

Introduction .....	i
<b>I. Le cours.....</b>	<b>1</b>
1. Échauffement : un peu d’algorithmique .....	1
2. Un peu plus d’algorithmique .....	3
3. Introduction à ncurses .....	5
3.1. Qu’est ce que Ncurses ?.....	5
3.2. Le minimum pour utiliser ncurses .....	5
3.3. Positionnement à l’écran.....	5
3.4. Premier exemple .....	6
3.5. Exercices .....	6
4. Ncurses et le clavier.....	9
4.1. La méthode <b>getch()</b> .....	9
4.2. D’autres méthodes.....	9
4.3. Les codes de retour de <b>getch()</b> .....	9
4.4. Un exemple.....	10
4.5. Exercices .....	10
5. Tout se complique.....	13
6. Pour qui sont ces serpents ? <sup>1</sup> .....	15
6.1. Première version simpliste .....	15
6.2. Attention aux bords.....	15
6.3. Un vrai serpent.....	15
6.4. Grandir .....	15
6.5. Enfin. . . ..	15
6.6. Améliorations éventuelles .....	15
<b>II. Correction des exercices.....</b>	<b>17</b>
7. Un peu d’algorithmique, correction .....	17
8. Un peu plus d’algorithmique, correction.....	21
9. Introduction à Ncurses, correction.....	23
10. Ncurses et le clavier, correction .....	27
11. Tout se complique, correction.....	29
12. Pour qui sont ces serpents, correction .....	31
12.1. Première version simpliste .....	31
12.2. Attention aux bords.....	31
12.3. Un vrai serpent.....	33
12.4. Grandir .....	34
12.5. Enfin. . . ..	35
12.6. En prime .....	35
<b>III. Annexes .....</b>	<b>39</b>
A. Où trouver ce document ?.....	39
B. C’est quoi Python ?.....	41
C. D’autres sources d’information .....	43

---

1. « Pour qui sont ces serpents qui sifflent sur vos têtes ? » Racine, *Andromaque*, V, 8



## Introduction

Vous avez devant les yeux le support du cours donné aux étudiants de 1<sup>ère</sup> et 2<sup>ème</sup> années de licence MIASHS, au 2<sup>ème</sup> semestre de l'année 2004-2005, à l'Université Lille3-Charles de Gaulle.

Ces étudiants ont au préalable suivi un cours d'initiation à la programmation, également en Python, basé sur l'environnement du Robot<sup>1</sup>.

Le but de ce cours est d'abord de montrer qu'on peut programmer sans l'environnement du Robot<sup>2</sup>. On essaiera d'arriver à des notions les complexes possible. Pour ne pas « effaroucher » le public on se placera dans des cas d'exercices ludiques qui, tout en utilisant des notions simples, amèneront à se poser des problèmes non triviaux.

---

1. <http://www.grappa.univ-lille3.fr/~taquet/Python>  
2. <http://www.grappa.univ-lille3.fr/~taquet/Python>



# Chapitre 1. Échauffement : un peu d'algorithmique

Quelques exercices pour se remettre en route... Les corrigés se trouvent au Chapitre 7.

1. Afficher tous les entiers de 21 à 145.
2. Afficher 250 étoiles (« \* »).
3. Écrire un programme qui écrit 500 fois « *Je dois faire des sauvegardes régulières de mes fichiers.* »
4. Calculer la somme de tous les entiers de 21 à 145.
5. Calculer 35! (factorielle).
6. Afficher un triangle rectangle composé d'étoiles (« \* ») de 20 de côté :

```
*
**
***
****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
```

7. Écrire un programme qui affiche la table de multiplication par 13.
8. Écrire un programme qui affiche la table de multiplication totale de  $\{1, \dots, 10\}$  par  $\{1, \dots, 10\}$ .
9. Sachant que le prix d'une vache est de 9000 F et que celui d'un mouton est de 5000 F, écrire un programme qui demande le nombre de moutons et de vaches et qui affiche le prix du troupeau ainsi constitué.
10. Saisir deux horaires (heures/minutes/secondes), les afficher dans l'ordre croissant.
11. Saisir deux horaires en heures minutes, secondes ; calculer le temps écoulé entre les deux (en secondes).
12. Saisir un entier N, afficher la somme des N premiers entiers.
13. Je suis ligoté sur les rails en garre d'Arras. Il vous faut écrire un programme qui affiche un tableau me permettant de connaître l'heure à laquelle je serai déchiqueté par le train parti de la gare du Nord à 9h (il y a 170 km entre la gare du Nord et Arras). Le tableau présentera les différentes heures possibles pour toutes les vitesses de 100 km/h à 300km/h, par pas de 10km/h, les résultats étant arrondis à la minute inférieure. On y trouvera par exemple:

```
160 km/h 10h03
170 km/h 10h00
180 km/h 9h56
```

- a. Écrire une procédure **tchacatchac(v)** qui reçoit en paramètre la vitesse du train et qui affiche l'heure à laquelle je serai écrasé.
  - b. Écrire le programme utilisant cette procédure et qui affiche le tableau demandé.
14. Un permis de chasse à points remplace désormais le permis de chasse traditionnel. Chaque chasseur possède au départ un capital de 100 points. S'il tue une poule il perd 1 point, 3 points pour un chien, 5 points pour une vache et 10 points pour son meilleur ami. Le permis coûte 200 euros.  
Écrire une fonction **permisSup** qui admet en paramètres les nombres de victimes du chasseur et qui renvoie la somme qu'il aura à déboursier pour ses permis supplémentaires.  
Puis utiliser cette fonction dans un programme qui demande le nombre de victimes et qui affiche la somme qu'aura à déboursier le chasseur.

15. Un gardien de phare va aux toilettes 5 fois par jour. Les WC sont au rez-de-chaussée. Écrire une procédure **hauteurParcourue(nbMarches, hauteurMarche)** qui reçoit en paramètres le nombre de marches du phare et la hauteur de chaque marche (exprimée en centimètres) et qui affiche :

Pour  $x$  marches de  $y$  cm, il parcourt  $z$  mètres en une semaine.

On n'oubliera pas :

- qu'une semaine comporte 7 jours,
- qu'une fois en bas, le gardien doit remonter,
- la hauteur des marches est exprimée en cm et le résultat doit l'être en mètres.

## Chapitre 2. Un peu plus d'algorithmique

Les corrigés se trouvent au Chapitre 8.

1. Saisir une suite de nombres terminée par -1, qui ne fait pas partie de la liste à traiter ; afficher le double de chacun des nombres à mesure qu'ils sont saisis, afficher FIN quand c'est fini.
2. Saisir une suite de nombres terminée par -1, qui ne fait pas partie de la liste à traiter ; afficher leur somme quand c'est fini.
3. Saisir une suite de nombres terminée par -1, qui ne fait pas partie de la liste à traiter ; afficher le plus grand quand c'est fini.
4. Coefficients du binôme :
  - a. Écrire une fonction qui renvoie  $n!$ .
  - b. Utiliser la fonction précédente pour écrire une fonction qui renvoie

$$C_n^p = \frac{n!}{p!(n-p)!}$$

- c. Utiliser les fonctions précédentes dans un programme qui affiche les coefficients du binôme pour toutes les valeurs de  $n$  dans  $\{0,1,\dots,20\}$  :

$$C_n^0 \quad C_n^1 \quad C_n^2 \quad \dots \quad C_n^{n-2} \quad C_n^{n-1} \quad C_n^n$$

5. La suite de Fibonacci est définie par les relations suivantes :

- $F_0=0$ ,
- $F_1=1$ ,
- $F_n=F_{n-1} + F_{n-2}$  pour tout  $n>1$ .

Donc :

- $F_0=0$ ,
- $F_1=1$ ,
- $F_2=0+1=1$ ,
- $F_3=1+1=2$ ,
- $F_4=2+1=3$ ,
- $F_5=3+2=5$ ,
- $F_6=5+3=8$ ,
- $F_7=8+5=13$ ,
- etc. . .

Écrire un programme qui affiche les 50 premières valeurs de  $F_n$ .

6. Fibonacci, le retour : afficher les rapports

$$\frac{F_n}{F_{n-1}}$$

ainsi que leurs différences avec le nombre

$$\frac{\sqrt{5}+1}{2}$$



## Chapitre 3. Introduction à ncurses

### Résumé

`Ncurses` est une bibliothèque qui fournit des fonctions de définition de touches du clavier, de couleurs d'écran et permet l'affichage de plusieurs fenêtres sans recouvrement sur un terminal texte.

### 3.1. Qu'est ce que Ncurses ?

(Ce paragraphe est extrait de <http://linuxfocus.unixtech.be/Francais/March2002/article233.shtml>.)

Voulez vous que vos programmes disposent d'une interface couleur sur un terminal texte ? `Ncurses` est une bibliothèque qui offre ce type de fonctionnalité pour les terminaux en mode texte. `Ncurses` peut :

- utiliser tout l'écran si vous le souhaitez,
- créer et gérer des fenêtres,
- utiliser 8 couleurs différentes,
- permettre le contrôle de votre programme par la souris,
- utiliser les touches de fonction du clavier.

Tout système UNIX conforme à la norme ANSI/POSIX supporte la bibliothèque `Ncurses`. De ce fait, elle est capable de détecter les propriétés du terminal à partir de la base de données du système, et donc de fournir une interface indépendante du terminal. Ainsi, `ncurses` est utilisable et fiable pour des travaux devant fonctionner sur différentes plates-formes et différents types de terminaux.

`Ncurses` est disponible à cette adresse : <http://www.gnu.org/software/ncurses/>.

### 3.2. Le minimum pour utiliser ncurses

Commençons par expliquer le minimum à mettre dans vos programmes pour qu'ils puissent utiliser `ncurses` (qui s'appelle d'ailleurs `curses`, sans « n », dans Python, mais ça ne change rien).

Voici le programme minimum :

```
import curses          # Importation de la bibliothèque curses
ecran = curses.initscr() # Créer une structure ecran

.....                # Votre programme se place là

curses.endwin()       # Terminer proprement
```

#### Remarque

Le nom **ecran** choisi ci-dessus est totalement arbitraire. Vous pouvez choisir ce que vous voulez. Ce n'est qu'un nom de variable. Mais il faudra adapter dans la suite des exemples.

On verra au fur et à mesure de l'avancée dans le cours d'autres commandes nécessaires au bon fonctionnement de `curses`.

#### Remarque

L'utilisation de `curses` modifie énormément le fonctionnement du terminal dans lequel vous exécutez le programme. La dernière ligne permet de laisser le terminal dans un état propre au sortir du programme.

Mais encore faut-il arriver à cette dernière ligne... Si votre programme *plante* il n'atteindra pas cette ligne et vous vous retrouverez à utiliser un terminal en très mauvais état...

Si cela vous arrive la solution est simple : tapez en aveugle la commande **reset** (suivie d'un appui sur la touche *Entrée*).

### 3.3. Positionnement à l'écran

Selon ce que vous voulez placer à l'écran vous utiliserez les procédures suivantes :

**addstr()** :

Pour positionner une chaîne de caractères .

Ses paramètres peuvent être :

**str** ou **ch** :

Affiche la chaîne **str** ou le caractère **ch** à la position actuelle du curseur.

**str** ou **ch**, **attr** :

Affiche la chaîne **str** ou le caractère **ch** à la position actuelle du curseur, en utilisant l'attribut **attr**.

**y**, **x**, **str** ou **ch** :

Affiche la chaîne **str** ou le caractère **ch** à la  $y^{\text{ème}}$  ligne et à la  $x^{\text{ème}}$  colonne.

**y**, **x**, **str** ou **ch**, **attr** :

Affiche la chaîne **str** ou le caractère **ch** à la  $y^{\text{ème}}$  ligne et à la  $x^{\text{ème}}$  colonne, en utilisant l'attribut **attr**.

Les attributs (**attr**) permettent de contrôler l'apparence de ce qui est affiché (vidéo inversée, gras, souligné, etc.).

Quand il est mentionné « position actuelle du curseur », il s'agit de la dernière position où on a fait un affichage. Cette position peut cependant être modifiée par la méthode **move(y,x)**.

**addch()** :

Se comporte exactement comme **addstr**, mais sert à positionner un caractère uniquement, et non une chaîne.

Ce caractère peut-être soit une chaîne Python de longueur 1, ou un entier, qui représente le code ASCII du caractère à afficher.

### 3.4. Premier exemple

Le programme suivant va placer la phrase « ça marche! » à la position  $(x,y)=(12,5)$ .

```
# -*- coding: utf-8 -*-
import curses          # Importation de la bibliothèque curses
ecran = curses.initscr() # Créer une structure ecran
ecran.addstr(5,12,"ça marche")
ecran.getch()         # Attente frappe clavier
curses.endwin()      # Terminer proprement
```

### 3.5. Exercices

Les corrigés se trouvent au Chapitre 9.

1. La position en haut à gauche est (0,0). Placez-y une étoile (« \* »).
2. La taille de l'écran s'obtient par la méthode **getmaxyx()** qui renvoie un couple de valeurs. Placez une étoile dans chaque coin de l'écran.
3. Placez une ligne horizontale de 20 étoiles au centre de l'écran.
4. Placez un rectangle composé d'étoiles, 20 colonnes sur 10 lignes, au centre de l'écran.

5. Même exercice que le précédent, mais le rectangle est creux (on ne voit que les bords).
6. Dessinez un disque composé d'étoiles, de rayon 10, au centre de l'écran.

Rappels :

- Un disque de rayon  $R$  est l'ensemble des points du plan dont la distance au centre du cercle est inférieure ou égale à  $R$ .
- La distance entre deux points de coordonnées  $(a,b)$  et  $(c,d)$  est la racine carrée de  $(a-c)^2+(b-d)^2$ .

7. Amélioration : tracez seulement le cercle (pas l'intérieur du disque).
8. Variation : tracez le plus grand cercle qu'il soit possible d'inscrire dans l'écran.



## Chapitre 4. `Ncurses` et le clavier

### 4.1. La méthode `getch()`

La façon la plus simple de permettre à un programme utilisant `Ncurses` de tenir compte des frappes au clavier est d'utiliser la méthode `getch()` : elle renvoie comme résultat le code ASCII (entre 0 et 255) du caractère correspondant à la touche frappée.

Les codes supérieurs à 255 correspondent aux touches *spéciales* (touches de fonctions, flèches, etc.). Voir Section 4.3 pour les principaux codes correspondant à ces touches.

### 4.2. D'autres méthodes

D'autres méthodes peuvent être utiles :

**`nodelay()`** :

Si on n'utilise pas cette fonction, le programme s'arrête quand il rencontre `getch()` et attend la frappe au clavier pour reprendre.

On l'utilise avec un paramètre :

**1** ou **True** :

Pour supprimer l'attente : le programme se contente de scruter le clavier et il continue, qu'une touche soit enfoncée ou non (la valeur renvoyée quand il n'y a pas de touche enfoncée est **ERR**, c'est-à-dire **-1**).

**0** ou **False** :

Pour activer l'attente.

C'est l'état par défaut.

**`clear()`** :

Efface l'écran.

**`echo()`** et **`noecho()`** :

La fonction `echo()` (resp. `noecho()`) active (resp. inhibe) l'écho à l'écran de la touche frappée.

**`keypad()`** :

Permet de transformer l'appui sur une touche de fonction en un seul code de caractère.

On l'utilise avec un paramètre :

**1** ou **True** :

L'utilisation d'une touche de fonction correspond à un seul code caractère. Voir Section 4.3 pour les principaux codes correspondant à ces touches.

**0** ou **False** :

L'utilisation d'une touche de fonction correspond à l'envoi d'une suite de plusieurs codes caractères, beaucoup plus difficile à interpréter.

C'est l'état par défaut.

### 4.3. Les codes de retour de `getch()`

Les codes renvoyés par `getch()` lors de l'appui d'une touche sont des entiers. Il est assez difficile de se souvenir de tous ces nombres. C'est pour cela qu'on utilise des constantes dont les noms sont choisis pour être facilement mémorisables.

Pour les utiliser dans un programme on écrira par exemple « `curses.KEY_F2` » pour désigner la touche *F2*.

En voici quelques exemples. Pour une liste plus complète voir, entre autres, *Curses, constants*<sup>5</sup>. (D'autres liens se trouvent Annexe C.)

Code	Signification
<code>KEY_DOWN</code>	Flèche bas
<code>KEY_UP</code>	Flèche haut
<code>KEY_LEFT</code>	Flèche gauche
<code>KEY_RIGHT</code>	Flèche droite
<code>KEY_Fn</code>	Touche de fonction <i>F<sub>n</sub></i> (valeurs possibles de <i>n</i> : de 1 à 64)

### 4.4. Un exemple

```
while 1:
    c = ecran.getch()
    if c == ord('p'): ecran.addstr(10,10,"Touche P")
    elif c == ord('q'): break # Sort de la boucle
    elif c == curses.KEY_HOME: ecran.clear()
```

Ce que fait cet extrait de programme :

- Il s'agit d'une boucle (potentiellement) sans fin (**while 1**).
- Si on appuie sur la touche **p**, il s'écrit « Touche P » à l'écran.
- Si on appuie sur la touche **q**, on sort de la boucle.
- Si on appuie sur la touche *Home*, on efface l'écran.
- Tout autre touche est sans effet.

### 4.5. Exercices

Les corrigés se trouvent au Chapitre 10.

- Écrire un programme qui commence par afficher une étoile en haut à gauche de l'écran, puis la déplace vers le bas à droite (direction Sud Est) à chaque fois qu'on appuie sur une touche.  
On sortira du programme en appuyant sur la touche « **q** ».  
On ne prendra pas la peine de contrôler la sortie de l'écran, on s'occupera de cela plus tard : si l'affichage sort, le programme plante ; ce n'est pas important pour l'instant. Mais dans ce cas n'oubliez de taper « **reset** » à l'*aveugle* pour retrouver un terminal en bon état.
- Écrire un programme qui affiche en alternance au même endroit sur l'écran l'une des deux phrases « Il marche » et « Il ne marche plus ». Le passage de l'un à l'autre se fera par l'appui sur la touche d'espacement, la sortie du programme se fera proprement par l'appui sur la touche « **q** ».
- Écrire un programme qui déplace le curseur à l'écran en utilisant les flèches du clavier.  
On ne prendra pas la peine de contrôler la sortie de l'écran dans cette première version.

5. <http://www.oopweb.com/Python/Documents/Official/Volume/lib/node183.html>

4. Améliorez le programme précédent en contrôlant la sortie de l'écran : si l'utilisateur essaie de faire sortir le curseur, rien ne se passe.
5. Écrire un programme qui permet de contrôler au clavier un compteur. On sortira du programme par l'appui sur la touche « **q** ».

Les flèches haut et bas feront varier le compteur de 1 (en plus ou en moins), tandis que les touches *page suivante* et *page précédente* feront varier le compteur de 100.



## Chapitre 5. Tout se complique...

Nous allons dans ce chapitre devoir affronter des exercices plus conséquents...

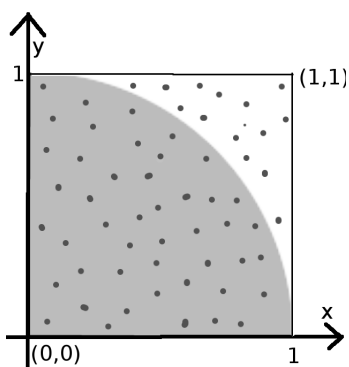
Les corrigés se trouvent au Chapitre 11.

1. Dessinez à l'écran à l'aide du clavier :

- On se déplace à l'écran avec les flèches.
- À tout moment l'appui de la touche d'espace change l'état de la case sur laquelle on est : si elle est occupée, elle devient vide ; si elle est vide, elle devient occupée (par exemple par une étoile, mais vous pouvez choisir autre chose).
- On sort du programme par l'appui sur la touche « **q** ».

2. Utilisation de la méthode de Monte-Carlo pour une évaluation de la valeur de  $\pi$ . Ou comment utiliser le hasard pour calculer  $\pi$ .

On répète un tirage aléatoire des coordonnées d'un point dans un carré de 1 de côté (voir image ci-dessous). On compte à la fois le nombre de tirages faits, et le nombre de fois où le point obtenu se trouve à l'intérieur du quart de disque de rayon 1 centré en  $(0,0)$ , représenté en gris dans l'image ci-dessous.



Quel rapport avec  $\pi$  direz-vous.

Il est immédiat. L'aire du disque de rayon 1 est  $\pi R^2$ . Donc celle du quart de disque est de  $\pi/4$ . Tandis que l'aire du carré est de 1.

Si les tirages sont uniformément répartis<sup>1</sup>, le rapport entre le nombre de points dans le quart de disque et le nombre de tirages doit être  $\pi/4$ .

Il suffit de multiplier ce rapport par 4 pour obtenir une approximation de  $\pi$ . Aussi simple que ça.

Eh bien maintenant, au boulot ! Il faudra que votre affichage ressemble à ceci :

```
728 tirages
pi=3.18131868132
ecart=0.0397260277289
```

À chaque nouveau tirage, l'affichage est mis à jour.

Rappels (?) :

- Pour avoir la « vraie » valeur de  $\pi$ , il faut utiliser **math.pi**, en ayant d'abord importé la bibliothèque **math**.
- Pour avoir une valeur aléatoire entre 0 et 1, il faut utiliser **random.random()**, en ayant d'abord importé la bibliothèque **random**.
- La distance du point de coordonnées  $(x,y)$  au point de coordonnées  $(0,0)$  est la racine carrée de  $x^2+y^2$ . Donc pour être dans le disque il suffit que  $x^2+y^2 < 1$

1. Ce qui, reconnaissons le, est loin d'être garanti. Mais on fera semblant d'y croire...



## Chapitre 6. Pour qui sont ces serpents ?<sup>1</sup>

Les corrigés se trouvent au Chapitre 12.

Nous allons dans ce chapitre réaliser un jeu simple, mais qui fait partie de l'histoire des jeux vidéo... (Certains diront peut-être préhistoire !). Il s'agit de *centipede*.

Vous pourrez trouver sur le web beaucoup de détails sur ce jeu. Mais en voici quelques mots en attendant : il s'agit du déplacement d'une sorte de serpent<sup>2</sup>. Ce serpent se déplace dans un espace clos. La tête en se déplaçant ne doit jamais recouper le reste du corps. Tout serait très simple si la longueur du serpent n'augmentait pas continuellement, rendant les manœuvres de plus en plus difficiles...

### 6.1. Première version simpliste

Dans cette première version le serpent n'aura pas de corps. On déplacera donc seulement sa tête.

Le mode de déplacement de la tête est simple :

- la tête est en déplacement continu, droit devant elle ;
- l'utilisateur ne peut que la faire tourner à droite ou à gauche ;

Les changements de direction se feront par appui sur les flèches du clavier.

Dans cette première version on ne cherchera pas à vérifier que la tête ne touche pas les bords de l'écran. Si cela arrive le programme *plante*, tant pis !

### 6.2. Attention aux bords

Dans cette deuxième version on fera attention à ne pas percuter les bords.

Pour cela on commencera par matérialiser les bords autour de l'écran.

On fera en sorte de faire afficher un message annonçant à l'utilisateur qu'il a perdu en cas de contact avec les bords (et le jeu s'arrêtera *proprement* bien entendu).

### 6.3. Un vrai serpent

Nous allons commencer à donner son apparence définitive au serpent : il ne faut plus se contenter d'une tête ; il faut lui donner un corps.

### 6.4. Grandir

Étape suivante : le serpent grandit avec le temps qui passe, par exemple d'une cellule tous les 5 tours (ou plus, ou moins, à vous de choisir...).

### 6.5. Enfin...

Enfin la dernière modification.

Votre programme doit vérifier que le serpent ne se *mord pas la queue*. La tête ne doit jamais recouper le corps.

Si cela arrivait, perdu !

---

1. « Pour qui sont ces serpents qui sifflent sur vos têtes ? » Racine, *Andromaque*, V, 8

2. En fait *centipede* est le mot anglais qui signifie *mille pattes*.

## **6.6. Améliorations éventuelles**

Vous pouvez prévoir de décompter le temps, d'ajouter des obstacles, d'inclure de la nourriture pour votre serpent, etc.

Ces modifications ne seront pas corrigées.

## Chapitre 7. Un peu d'algorithmique, correction

Correction des exercices du Chapitre 1.

1.

```
for i in range (21,146):  
    print i,
```

2.

```
for i in range (1,251):  
    print '*',
```

On aurait aussi pu écrire :

```
print 250*'*'
```

3.

```
for i in range (1,501):  
    print i,'Je dois faire des sauvegardes régulières de mes fichiers'
```

4.

```
s = 0  
for i in range (21,146):  
    s += i  
print 'somme =',s
```

5.

```
f = 1  
for i in range (1,36):  
    f *= i  
print '36! =',f
```

6.

```
for i in range (1,21):  
    for j in range (1,i+1):  
        print '*',  
    print
```

On aurait aussi pu écrire :

```
for i in range (1,21):  
    print i*'*'
```

7.

```
for i in range (1,14):  
    print i,'x 13 =',i*13
```

8.

```
for i in range (1,11):  
    for j in range (1,11):
```

```
        print i*j, '\t',
    print
```

ou, mieux :

```
print '\t ',
for j in range (1,11):
    print j, '\t ',
print
print '\t',
for j in range (1,11):
    print '-----\t',
print
for i in range (1,11):
    print ' ', i, '\t|',
    for j in range (1,11):
        print i*j, '\t ',
    print
```

9.

```
nbMoutons = input(" Entrez le nombre de moutons : ")
nbVaches = input(" Entrez le nombre de vaches : ")
valeur = nbVaches * 9000 + nbMoutons * 5000
print "La valeur du troupeau est de ", valeur
```

10.

```
def saisie (texte):
    h = input("Horaire n%s.   heures : " %texte)
    m = input("                minutes : ")
    s = input("                secondes : ")
    return h,m,s

def convertir(h,m,s):
    return h*3600+m*60+s

h1,m1,s1 = saisie (1)
h2,m2,s2 = saisie (2)
horaire1 = convertir(h1,m1,s1)
horaire2 = convertir(h2,m2,s2)
if horaire1 < horaire2:
    print h1, 'h', m1, 'mn', s1, 's est avant ', h2, 'h', m2, 'mn', s2, 's'
else:
    print h2, 'h', m2, 'mn', s2, 's est avant ', h1, 'h', m1, 'mn', s1, 's'
```

11.

```
#!/usr/bin/env python
#-*- coding: utf-8
def saisie (texte):
    h = input("Horaire n%s.   heures : " %texte)
    m = input("                minutes : ")
    s = input("                secondes : ")
    return h,m,s

def convertir(h,m,s):
    return h*3600+m*60+s

h1,m1,s1 = saisie (1)
h2,m2,s2 = saisie (2)
horaire1 = convertir(h1,m1,s1)
horaire2 = convertir(h2,m2,s2)
print 'durée écoulée : '
if horaire1 < horaire2:
    print horaire2-horaire1
else:
```

```
print horaire1-horaire2
```

ou mieux :

```
#-*- coding: utf-8
def saisie (texte):
    h = input("Horaire n°s. heures : " %texte)
    m = input("          minutes : ")
    s = input("          secondes : ")
    return h,m,s

def convertir(h,m,s):
    return h*3600+m*60+s

h1,m1,s1 = saisie (1)
h2,m2,s2 = saisie (2)
print 'durée écoulée : ',abs( convertir(h1,m1,s1) - convertir(h2,m2,s2) )
```

12.

```
n = input("un entier SVP : ")
s = 0
for i in range (1,n+1):
    s += i
print 'La somme est',s
```

ou :

```
n = input("un entier SVP : ")
print 'La somme est',n*(n+1)/2
```

13.

```
#-*- coding: utf-8
def tchacatchac(v):
    heure = 9 + int(170 / v)
    minutes = (60 * 170 / v) % 60
    print " À", v, "km/h, je me fais déchiqueter à ", heure, "h", minutes, "mn."

i = 100
while i <= 300:
    tchacatchac(i)
    i += 10
```

14.

```
#-*- coding: utf-8
def PermisSup(p,c,v,a):
    pointsperdus = p+3*c+5*v+10*a
    nbrepermis = pointsperdus/100
    return 200*nbrepermis

poules = input("Combien de poules ? ")
chiens = input("Combien de chiens ? ")
vaches = input("Combien de vaches ? ")
amis = input(" Combien d'amis ? ")
print 'À payer : ',
payer = PermisSup(poules,chiens,vaches,amis)
if payer == 0:
    print "rien à payer"
else:
    print payer,'euros'
```

15.

## Chapitre 7. Un peu d'algorithmique, correction

```
#-*- coding: utf-8
def hauteurParcourue (nb,h):
    print "Pour",nb,"marches de",h,"cm, il parcourt",
    print nb*h*2*5*7/100,"mètres en une semaine"

nbmarches = input("Combien de marches ? ")
hauteurmarche = input("Hauteur (en centimètres) d'une marche ? ")
hauteurParcourue (nbmarches,hauteurmarche)
```

## Chapitre 8. Un peu plus d'algorithmique, correction

Correction des exercices du Chapitre 2.

1.

```
i = input ("Entrez votre entier : ")
while i <> -1:
    print "2 * ", i, " = ", 2*i
    i = input ("Entrez un autre entier : ")
print "FIN."
```

2.

```
i = input ("Entrez votre entier : ")
s = 0
while i <> -1:
    s += i
    i = input ("Entrez un autre entier : ")
print "La somme est :",s
```

3.

```
n = input( "Entrez un entier :")
max = n
while n <> -1:
    if n > max:
        max = n
    n = input ( "Entrez un autre entier : ")
print "L'entier le plus grand de la suite est :",max
```

4.

```
def fact(n):
    f = 1
    for i in range(1,n+1):
        f *= i
    return f

def cnp(n,p):
    return fact(n)/(fact(p)*fact(n-p))

def coefficients(n):
    for i in range(0,n+1):
        print cnp(n,i),
    print

for i in range(0,21):
    coefficients(i)
```

5.

```
def fib(n):
    if n == 0: return 0
    if n == 1: return 1
    moins2 = 0
    moins1 = 1
    for i in range(2,n+1):
        f = moins1+moins2
```

```
        moins2 = moins1
        moins1 = f
    return f

for i in range(0,50):
    print i, '\t', fib(i)
```

6.

```
from math import sqrt

def fib(n):
    if n == 0: return 0
    if n == 1: return 1
    moins2 = 0
    moins1 = 1
    for i in range(2,n+1):
        f = moins1+moins2
        moins2 = moins1
        moins1 = f
    return f

nbor = (sqrt(5)+1)/2
print "nombre d'or :",nbor
print
for i in range(2,50):
    x = 1.0*fib(i)/fib(i-1)
    print i, '\t', x, '\t', x-nbor
```

## Chapitre 9. Introduction à Ncurses, correction

Correction des exercices du Chapitre 3.

1.

```
import curses
ecran = curses.initscr()
ecran.addch(0,0,"*")
ecran.getch()
curses.endwin()
```

2. Quand vous écrivez « **y, x = écran.getmaxyx()** » n'oubliez pas que **x** et **y** représentent les dimensions de l'écran. Comme le système de coordonnées commence à (0,0), on ne pourra donc atteindre que **x-1** et **y-1**.

```
import curses
ecran = curses.initscr()
y, x = écran.getmaxyx()
ecran.addch(0,0,"*")
ecran.addch(y-1,0,"*")
ecran.addch(0,x-1,"*")
ecran.addch(y-1,x-2,"*")
ecran.addch(y-2,x-1,"*")
ecran.getch()
curses.endwin()
```

Vous avez sans doute remarqué que la position en bas à droite ne peut pas être atteinte. C'est parce qu'après y avoir écrit le curseur sortirait de l'écran.

3.

```
import curses
ecran = curses.initscr()
y, x = écran.getmaxyx()
ecran.move(y/2,x/2-10)
for i in range (0,20):
    écran.addch("*")
ecran.move(0,0)
ecran.getch()
curses.endwin()
```

ou, mieux :

```
import curses
ecran = curses.initscr()
y, x = écran.getmaxyx()
ecran.addstr(y/2,x/2-10,20*"")
ecran.move(0,0)
ecran.getch()
curses.endwin()
```

4.

```
import curses
ecran = curses.initscr()
y, x = écran.getmaxyx()
for i in range (0,10):
    écran.addstr(y/2-5+i,x/2-10,20*"")
ecran.move(0,0)
ecran.getch()
curses.endwin()
```

5. En trichant :

```
import curses
ecran = curses.initscr()
y, x = ekran.getmaxyx()
for i in range (0,10):
    ekran.addstr(y/2-5+i,x/2-10,20*" *")
for i in range (0,8):
    ekran.addstr(y/2-4+i,x/2-9,18*" ")
ecran.move(0,0)
ecran.getch()
curses.endwin()
```

Mieux :

```
import curses
ecran = curses.initscr()
y, x = ekran.getmaxyx()
ecran.addstr(y/2-5,x/2-10,20*" *")
for i in range (1,9):
    ekran.addch(y/2-5+i,x/2-10,"*")
    ekran.addch(y/2-5+i,x/2+9,"*")
ecran.addstr(y/2+4,x/2-10,20*" *")
ecran.move(0,0)
ecran.getch()
curses.endwin()
```

6.

```
import curses
ecran = curses.initscr()
ymax, xmax = ekran.getmaxyx()
x0,y0 = xmax/2,ymax/2
for x in range (0,xmax):
    for y in range(0,ymax):
        d = (x0-x)*(x0-x)+(y0-y)*(y0-y)
        if d < 100:
            ekran.addch(y,x,"*")
ecran.move(0,0)
ecran.getch()
curses.endwin()
```

7.

```
import curses
ecran = curses.initscr()
ymax, xmax = ekran.getmaxyx()
x0,y0 = xmax/2,ymax/2
for x in range (0,xmax):
    for y in range(0,ymax):
        d = (x0-x)*(x0-x)+(y0-y)*(y0-y)
        if d <= 108 and d >= 92 :
            ekran.addch(y,x,"*")
ecran.move(0,0)
ecran.getch()
curses.endwin()
```

8.

```
import curses
ecran = curses.initscr()
ymax, xmax = ekran.getmaxyx()
rayon=min(xmax,ymax)/2-1
x0,y0 = xmax/2,ymax/2
inf=rayon*rayon-8
sup=rayon*rayon+8
```

```
for x in range (0,xmax):
    for y in range(0,ymax):
        d = (x0-x)*(x0-x)+(y0-y)*(y0-y)
        if d <= sup and d >= inf :
            ecran.addch(y,x,"*")
ecran.move(0,0)
ecran.getch()
curses.endwin()
```



## Chapitre 10. `ncurses` et le clavier, correction

Correction des exercices du Chapitre 4.

1.

```
import curses

ecran = curses.initscr()
curses.noecho()
c = ""
pos = 0;

while c<>ord('q') :
    écran.clear()
    écran.addch(pos,pos,'*')
    c = écran.getch()
    pos += 1

curses.echo()
curses.endwin()
```

2.

```
import curses

ecran = curses.initscr()
curses.noecho()
c = ""
etat = True

while c<>ord('q') :
    écran.clear()
    if etat :
        écran.addstr(10,10,'il marche')
    else :
        écran.addstr(10,10,'il ne marche plus')
    c = écran.getch()
    if c == ord(' ') :
        etat = not etat

curses.echo()
curses.endwin()
```

3.

```
import curses

ecran = curses.initscr()
curses.noecho()
écran.keypad(1)

écran.clear()
x = 20
y = 20
while 1 :
    écran.clear()
    écran.move(y,x)
    c = écran.getch()
    if c == curses.KEY_DOWN : y += 1
    elif c == curses.KEY_UP : y -= 1
    elif c == curses.KEY_LEFT : x -= 1
    elif c == curses.KEY_RIGHT : x += 1
    elif c == ord('q') : break

curses.echo()
curses.endwin()
```

4.

```
import curses

ecran = curses.initscr()
curses.noecho()
ecran.keypad(1)

ecran.clear()
ymax,xmax = ekran.getmaxyx()
x = 20
y = 20
while 1 :
    ekran.clear()
    if x > xmax-1 : x = xmax-1
    if x < 0 : x = 0
    if y > ymax-1 : y = ymax-1
    if y < 0 : y = 0
    ekran.move(y,x)
    c = ekran.getch()
    if c == curses.KEY_DOWN : y = y+1
    elif c == curses.KEY_UP : y = y-1
    elif c == curses.KEY_LEFT : x = x-1
    elif c == curses.KEY_RIGHT : x = x+1
    elif c == ord('q') : break

curses.echo()
curses.endwin()
```

5.

```
import curses

ecran = curses.initscr()
curses.noecho()
ecran.keypad(1)

ecran.clear()
compteur=0
while 1:
    ekran.clear()
    ekran.addstr(10,10,str(compteur))
    c = ekran.getch()
    if c == curses.KEY_DOWN: compteur -= 1
    elif c == curses.KEY_UP: compteur += 1
    elif c == curses.KEY_NPAGE: compteur -= 100
    elif c == curses.KEY_PPAGE: compteur += 100
    elif c == ord('q'): break

curses.echo()
curses.endwin()
```

# Chapitre 11. Tout se complique, correction

Correction des exercices du Chapitre 5.

1.

```
import curses

def deplacement(touche,x,y):
    if touche == curses.KEY_DOWN : y += 1
    elif touche == curses.KEY_UP : y -= 1
    elif touche == curses.KEY_LEFT : x -= 1
    elif touche == curses.KEY_RIGHT : x += 1
    return x,y

def controle(x,y):
    if x > xmax-1 : x = xmax-1
    if x < 0 : x=0
    if y > ymax-1 : y = ymax-1
    if y < 0 : y=0
    return x,y

def dessine (x,y):
    if ecran.inch(y,x) == ord(' '):
        ecran.addch (y,x,'*')
    else:
        ecran.addch (y,x,' ')
    ecran.move (y,x)

ecran = curses.initscr()
curses.noecho()
ecran.keypad(1)

ecran.clear()
ymax,xmax = ecran.getmaxyx()
x = 20
y = 20
while 1:
    x,y = controle (x,y)
    ecran.move(y,x)
    c = ecran.getch()
    if c == ord('q') : break
    elif c == ord(' ') : dessine (x,y)
    else : x,y = deplacement (c,x,y)

curses.echo()
curses.endwin()
```

On remarquera dans ce programme l'utilisation de procédures et fonctions :

- (1) pour le calcul de la nouvelle position du curseur après un éventuel appui sur une des flèches de déplacement ;
- (2) pour le contrôle de la position du curseur ; on l'empêche de sortir des limites imposées ;
- (3) pour changer le dessin de ce qu'il y a sous le curseur : étoile si rien, rien si étoile ;
- (4) corps du programme.

Elles ne sont pas là pour éviter de répéter le même code à plusieurs endroits du programme (chaque fonction ou procédure n'est utilisée qu'une fois), ce qui est une des principales raisons possibles de leur utilisation.

Ici elles sont utilisées pour rendre le code plus clair : le corps (4) du programme est plus facile à lire et à comprendre, car il est plus court.

Vous pouvez comparer avec la version suivante, sans fonction ni procédure, mono-bloc et plus difficile à analyser :

```
import curses

ecran = curses.initscr()
```

```
curses.noecho()
ecran.keypad(1)

ecran.clear()
ymax,xmax = ecran.getmaxyx()
x = 20
y = 20
while 1:
    if x > xmax-1 : x = xmax-1
    if x < 0 : x=0
    if y > ymax-1 : y = ymax-1
    if y < 0 : y=0
    ecran.move(y,x)
    c = ecran.getch()
    if c == ord('q') : break
    elif c == ord(' ') :
        if ecran.inch(y,x) == ord(' '):
            ecran.addch (y,x,'*')
        else:
            ecran.addch (y,x,' ')
            ecran.move (y,x)
    else :
        if c == curses.KEY_DOWN : y += 1
        elif c == curses.KEY_UP : y -= 1
        elif c == curses.KEY_LEFT : x -= 1
        elif c == curses.KEY_RIGHT : x += 1

curses.echo()
curses.endwin()
```

Cette version reste encore presque facile à comprendre (ce programme n'est pas très long), mais on commence à avoir du mal à y trouver une décomposition logique, alors que c'était immédiat dans la première version.

2.

```
import random
import curses
import math

ecran = curses.initscr()
curses.noecho()
ecran.nodelay(1)

tirages=0
disque=0

while 1:
    if ecran.getch() == ord('q') : break
    x = random.random()
    y = random.random()
    tirages += 1
    if x*x+y*y <= 1:
        disque += 1
    monpi = 4.0*disque/tirages
    ecran.clear()
    ecran.addstr(5,5,str(tirages)+" tirages")
    ecran.addstr(6,5,"pi="+str(monpi))
    ecran.addstr(7,5,"ecart="+str(abs(monpi-math.pi)))

curses.echo()
curses.endwin()
```

# Chapitre 12. Pour qui sont ces serpents, correction

Correction des exercices du Chapitre 6.

## 12.1. Première version simpliste

```
#-*- coding: utf-8 -*-
import curses # pour le dessin
import time   # pour la mesure du temps d'attente

# renvoie la nouvelle direction après appui sur la touche "DROITE"
def droite() :
    if direction == DROITE : return BAS
    elif direction == BAS : return GAUCHE
    elif direction == GAUCHE : return HAUT
    else : return DROITE

# renvoie la nouvelle direction après appui sur la touche "GAUCHE"
def gauche() :
    if direction == DROITE : return HAUT
    elif direction == HAUT : return GAUCHE
    elif direction == GAUCHE : return BAS
    else : return DROITE

# calcule la nouvelle position et place le serpent
def deplace() :
    l = ligne + direction[LGN]
    c = colonne + direction[COL]
    ecran.addch(l,c,'O')
    return l,c

ecran = curses.initscr() # initialisation de l'écran
ecran.nodelay(True)     # pas d'attente sur getch()
ecran.keypad(True)      # UN code sur les touches de fonctions
curses.curs_set(False)  # cacher le curseur

# définitions des directions de déplacement
DROITE = (0,1)
GAUCHE = (0,-1)
HAUT = (-1,0)
BAS = (1,0)

# indices pour les accès aux éléments d'un couple de coordonnées
LGN = 0
COL = 1

# initialisations du positionnement du serpent
hauteur, largeur = ecran.getmaxyx() # taille de l'écran
ligne,colonne = hauteur/3,largeur/3 # position (de départ : arbitraire!)
direction = DROITE # direction (de départ : arbitraire!)

c="" # pour que c existe à la ligne suivante
while c <> ord('q'): # tant qu'on n'a pas appuyé sur la touche 'q'
    ecran.clear() # effacer l'écran
    ligne,colonne = deplace() # calcule la nouvelle position et place le serpent
    time.sleep(0.1) # attente pour ralentir (arbitraire!)
    c=ecran.getch() # saisie de l'appui sur une touche du clavier
    if c == curses.KEY_LEFT: direction = gauche() # on tourne à gauche
    elif c == curses.KEY_RIGHT: direction = droite() # on tourne à droite

# remettre l'écran en "bon" état
curses.endwin() # sortie du mode "curses"
print "\n\n\tF I N I !\n\n" # message de fin!
```

## 12.2. Attention aux bords

```
#-*- coding: utf-8 -*-
import curses # pour le dessin
import time   # pour la mesure du temps d'attente

# renvoie la nouvelle direction après appui sur la touche "DROITE"
def droite() :
    if direction == DROITE : return BAS
    elif direction == BAS : return GAUCHE
    elif direction == GAUCHE : return HAUT
    else : return DROITE

# renvoie la nouvelle direction après appui sur la touche "GAUCHE"
def gauche() :
    if direction == DROITE : return HAUT
    elif direction == HAUT : return GAUCHE
    elif direction == GAUCHE : return BAS
    else : return DROITE

# calcule la nouvelle position et place le serpent
def deplace() :
    l = ligne + direction[LGN]
    c = colonne + direction[COL]
    ecran.addch(l,c,'O')
    return l,c

# =====
#                NOUVEAU
# =====
# teste si on a touché les bords
def touche() :
    return (ligne <= 0) or (colonne <= 0) or (ligne >= hauteur-1) or (colonne >= largeur-1)

# affiche un message (s) au centre de l'écran
def message(s) :
    vide = (largeur-len(s))/2 # longueur libre de chaque côté du message
    debut = hauteur/2 - 2    # hauteur d'affichage du message
    ecran.addstr(debut,0,largeur*"=") # un trait au-dessus
    ecran.addstr(debut+1,0,vide*" "+s+vide*" ") # message au centre de la ligne
    ecran.addstr(debut+2,0,largeur*"=") # un trait au-dessous
    ecran.nodelay(False) # attente sur les getch()
    ecran.getch() # pour attendre...

# dessine le bord du terrain
def bord():
    ecran.addstr(0,0,largeur*"=") # bord du haut
    ecran.addstr(hauteur-1,0,largeur*"=") # bord du bas
    for lg in range(1,hauteur-1): # les côtés :
        ecran.addstr(lg,0,"+") # à gauche
        ecran.addstr(lg,largeur-1,"+") # à droite
# =====

ecran = curses.initscr() # initialisation de l'écran
ecran.nodelay(True) # pas d'attente sur getch()
ecran.keypad(True) # UIN code sur les touches de fonctions
curses.curs_set(False) # cacher le curseur

# définitions des directions de déplacement
DROITE = (0,1)
GAUCHE = (0,-1)
HAUT = (-1,0)
BAS = (1,0)

# indices pour les accès aux éléments d'un couple de coordonnées
LGN = 0
COL = 1

# initialisations du positionnement du serpent
hauteur, largeur = ecran.getmaxyx() # taille de l'écran
```

```

ligne,colonne = hauteur/3,largeur/3 # position (de départ : arbitraire!)
direction = DROITE # directon (de départ : arbitraire!)
largeur -= 1 # NOUVEAU : pour ne pas accéder à la dernière colonne
# (bord : pb d'affichage dans coin en bas à droite)

c="" # pour que c existe à la ligne suivante
while c <> ord('q'): # tant qu'on n'a pas appuyé sur la touche 'q'
    ecran.clear() # effacer l'écran
    bord() # dessin du bord du terrain
    ligne,colonne = deplace() # calcule la nouvelle position et place le serpent
    if touche() : # si on touche le bord ...
        message ('Perdu !') # ... affichage du message "Perdu!"
        break # ... sortie de la boucle while (donc fin du programme)
    time.sleep(0.1) # attente pour ralentir (arbitraire!)
    c=ecran.getch() # saisie de l'appui sur une touche du clavier
    if c == curses.KEY_LEFT: direction = gauche()
        # on tourne à gauche
    elif c == curses.KEY_RIGHT: direction = droite()
        # on tourne à droite

# remettre l'écran en "bon" état
curses.endwin() # sortie du mode "curses"
print "\n\n\tF I N I !\n\n" # message de fin!

```

### 12.3. Un vrai serpent

```

# -*- coding: utf-8 -*-
import curses # pour le dessin
import time # pour la mesure du temps d'attente

# renvoie la nouvelle direction après appui sur la touche "DROITE"
def droite() :
    if direction == DROITE : return BAS
    elif direction == BAS : return GAUCHE
    elif direction == GAUCHE : return HAUT
    else : return DROITE

# renvoie la nouvelle direction après appui sur la touche "GAUCHE"
def gauche() :
    if direction == DROITE : return HAUT
    elif direction == HAUT : return GAUCHE
    elif direction == GAUCHE : return BAS
    else : return DROITE

# calcule la nouvelle position et place le serpent
def deplace() :
    serpent.append((ligne,colonne)) # NOUVEAU : ajoute la position actuelle de
    # la tête en fin de la liste des positions

    l = ligne + direction[LGN]
    c = colonne + direction[COL]
    for z in serpent: # NOUVEAU : pour toutes les cases du serpent ...
        ecran.addch(z[LGN],z[COL], '*') # ... afficher la case
    ecran.addch(l,c,'O')
    return l,c

# teste si on a touché les bords
def touche() :
    return (ligne <= 0) or (colonne <= 0) or (ligne >= hauteur-1) or (colonne >= largeur-1)

# affiche un message (s) au centre de l'écran
def message(s) :
    vide = (largeur-len(s))/2 # longueur libre de chaque côté du message
    debut = hauteur/2 - 2 # hauteur d'affichage du message
    ecran.addstr(debut ,0,largeur*"=") # un trait au-dessus
    ecran.addstr(debut+1,0,vide*" "+s+vide*" ") # message au centre de la ligne
    ecran.addstr(debut+2,0,largeur*"=") # un trait au-dessous
    ecran.nodelay(False) # attente sur les getch()

```

## Chapitre 12. Pour qui sont ces serpents, correction

```
    ecran.getch() # pour attendre...

# dessine le bord du terrain
def bord():
    ecran.addstr(0,0,largeur*"+") # bord du haut
    ecran.addstr(hauteur-1,0,largeur*"+") # bord du bas
    for lg in range(1,hauteur-1): # les côtés ...
        ecran.addstr(lg,0,"+") # ... à gauche
        ecran.addstr(lg,largeur-1,"+") # ... à droite

ecran = curses.initscr() # initialisation de l'écran
ecran.nodelay(True) # pas d'attente sur getch()
ecran.keypad(True) # UN code sur les touches de fonctions
curses.curs_set(False) # cacher le curseur
serpent = [] # création du serpent (liste) vide
longueur = 15 # taille du serpent

# définitions des directions de déplacement
DROITE = (0,1)
GAUCHE = (0,-1)
HAUT = (-1,0)
BAS = (1,0)

# indices pour les accès aux éléments d'un couple de coordonnées
LGN = 0
COL = 1

# initialisations du positionnement du serpent
hauteur, largeur = ecran.getmaxyx() # taille de l'écran
ligne,colonne = hauteur/3,largeur/3 # position (de départ : arbitraire!)
direction = DROITE # direction (de départ : arbitraire!)
largeur -= 1 # pour ne pas accéder à la dernière colonne
# (bord : pb d'affichage dans coin en bas à droite)

c="" # pour que c existe à la ligne suivante
while c <> ord('q'): # tant qu'on n'a pas appuyé sur la touche 'q'
    ecran.clear() # effacer l'écran
    bord() # dessin du bord du terrain
    ligne,colonne = deplace() # calcule la nouvelle position et place le serpent
    serpent = serpent[-longueur:] # NOUVEAU : ne garde que la fin du serpent (taille invariable)
    if touche(): # si on touche le bord ...
        message('Perdu !') # ... affichage du message "Perdu!"
        break # ... sortie de la boucle while (donc fin du programme)
    time.sleep(0.1) # attente pour ralentir (arbitraire!)
    c=ecran.getch() # saisie de l'appui sur une touche du clavier
    if c == curses.KEY_LEFT: direction = gauche() # on tourne à gauche
    elif c == curses.KEY_RIGHT: direction = droite() # on tourne à droite

# remettre l'écran en "bon" état
curses.endwin() # sortie du mode "curses"
print "\n\n\tF I N I !\n\n" # message de fin!
```

### 12.4. Grandir

Peu de changement entre ces deux versions. Vous trouverez ci-dessous une partie du programme, celle qui contient les changements (les lignes sont indiquées par un commentaire).

```
# initialisations du positionnement du serpent
hauteur, largeur = ecran.getmaxyx() # taille de l'écran
ligne,colonne = hauteur/3,largeur/3 # position (de départ : arbitraire!)
direction = DROITE # direction (de départ : arbitraire!)
largeur -= 1 # pour ne pas accéder à la dernière colonne
# (bord : pb d'affichage dans coin en bas à droite)
```

```

compteur=0                                # NOUVEAU : compteur entre 2 croissance du serpent

c=""                                       # pour que c existe à la ligne suivante
while c <> ord('q'):                       # tant qu'on n'a pas appuyé sur la touche 'q'
    ecran.clear()                          # effacer l'écran
    bord()                                 # dessin du bord du terrain
    ligne,colonne = deplace()              # calcule la nouvelle position et place le serpent
    serpent = serpent[-longueur:]         # ne garde que la fin du serpent (taille invariable)
    compteur += 1                           # NOUVEAU : un tour de plus
    if compteur > 5:                        # NOUVEAU : si on passe 5 tours ...
        compteur = 0                       # NOUVEAU : ... remise du compteur à zéro
        longueur += 1                      # NOUVEAU : ... fait grandir le serpent
    if touche() :                           # si on touche le bord ...
        message ('Perdu !')                # ... affichage du message "Perdu!"
        break                               # ... sortie de la boucle while (donc fin du programme)

```

## 12.5. Enfin...

Il suffit de modifier légèrement la fonction `toucher()`.

```

# NOUVEAU : la fonction est réécrite entièrement
# teste si on a touché les bords ou le corps
def touche() :
    # si la tête du serpent touche le bord
    if (ligne <= 0) or (colonne <= 0) or (ligne >= hauteur-1) or (colonne >= largeur-1) :
        return True # renvoyer VRAI (= on a perdu)
    # pour toutes les cases du serpent
    for z in serpent:
        if z == (ligne, colonne) : # si la case est au même endroit que la tête ...
            return True # ... renvoyer VRAI (= on a perdu)
    # si on arrive ici, c'est qu'il n'y a pas eu de problème
    return False # renvoyer FAUX (= pas de problème)

```

## 12.6. En prime

En cadeau une version complète, légèrement modifiée, qui utilise différemment les flèches de direction. L'appui sur une des quatre flèches de direction fait tourner le serpent dans cette direction de façon absolue : par exemple, si on appuie sur la flèche *HAUT* le serpent part vers le haut.

```

# -*- coding: utf-8 -*-
import curses # pour le dessin
import time   # pour la mesure du temps d'attente

# renvoie la nouvelle direction après appui sur la touche ""
def bas() :
    if (direction == DROITE) or (direction==GAUCHE) :
        return BAS
    return direction

# renvoie la nouvelle direction après appui sur la touche "haut"
def haut() :
    if (direction == DROITE) or (direction==GAUCHE) :
        return HAUT
    return direction

# renvoie la nouvelle direction après appui sur la touche "droite"
def droite() :
    if (direction == HAUT) or (direction==BAS) :
        return DROITE
    return direction

# renvoie la nouvelle direction après appui sur la touche "gauche"
def gauche() :
    if (direction == HAUT) or (direction==BAS) :

```

## Chapitre 12. Pour qui sont ces serpents, correction

```
        return GAUCHE
    return direction

# calcule la nouvelle position et place le serpent
def deplace() :
    serpent.append((ligne, colonne)) # ajoute la position actuelle de la tête
                                     # en fin de la liste des positions

    l = ligne + direction[LGN]
    c = colonne + direction[COL]
    for z in serpent: # pour toutes les cases du serpent ...
        ecran.addch(z[LGN], z[COL], ' * ') # ... afficher la case
    ecran.addch(l, c, 'O')
    return l, c

# NOUVEAU : la fonction est réécrite entièrement
# teste si on a touché les bords ou le corps
def touche() :
    # si la tête du serpent touche le bord
    if (ligne <= 0) or (colonne <= 0) or (ligne >= hauteur-1) or (colonne >= largeur-1) :
        return True # renvoyer VRAI (= on a perdu)
    # pour toutes les cases du serpent
    for z in serpent:
        if z == (ligne, colonne) : # si la case est au même endroit que la tête ...
            return True # ... renvoyer VRAI (= on a perdu)
    # si on arrive ici, c'est qu'il n'y a pas eu de problème
    return False # renvoyer FAUX (= pas de problème)

# affiche un message (s) au centre de l'écran
def message(s) :
    vide = (largeur-len(s))/2 # longueur libre de chaque côté du message
    debut = hauteur/2 - 2 # hauteur d'affichage du message
    ecran.addstr(debut, 0, largeur*"=") # un trait au-dessus
    ecran.addstr(debut+1, 0, vide*" "+s+vide*" ") # message au centre de la ligne
    ecran.addstr(debut+2, 0, largeur*"=") # un trait au-dessous
    ecran.nodelay(False) # attente sur les getch()
    ecran.getch() # pour attendre...

# dessine le bord du terrain
def bord():
    ecran.addstr(0, 0, largeur*"+" ) # bord du haut
    ecran.addstr(hauteur-1, 0, largeur*"+" ) # bord du bas
    for lg in range(1, hauteur-1): # les côtés ...
        ecran.addstr(lg, 0, "+" ) # ... à gauche
        ecran.addstr(lg, largeur-1, "+" ) # ... à droite

ecran = curses.initscr() # initialisation de l'écran
ecran.nodelay(True) # pas d'attente sur getch()
ecran.keypad(True) # UN code sur les touches de fonctions
curses.curs_set(False) # cacher le curseur
serpent = [] # création du serpent (liste) vide
longueur = 15 # taille du serpent

# définitions des directions de déplacement
DROITE = (0, 1)
GAUCHE = (0, -1)
HAUT = (-1, 0)
BAS = (1, 0)

# indices pour les accès aux éléments d'un couple de coordonnées
LGN = 0
COL = 1

# initialisations du positionnement du serpent
hauteur, largeur = ecran.getmaxyx() # taille de l'écran
ligne, colonne = hauteur/3, largeur/3 # position (de départ : arbitraire!)
direction = DROITE # direction (de départ : arbitraire!)
largeur -= 1 # pour ne pas accéder à la dernière colonne
# (bord : pb d'affichage dans coin en bas à droite)

compteur=0 # compteur entre 2 croissance du serpent
```

```

c=""
while c <> ord('q'):
    ecran.clear()
    bord()
    ligne,colonne = deplace()
    serpent = serpent[-longueur:]
    compteur += 1
    if compteur > 5:
        compteur = 0
        longueur += 1
    if touche():
        message('Perdu !')
        break
    time.sleep(0.1)
    c=ecran.getch()
    if c == curses.KEY_LEFT: direction = gauche()
    elif c == curses.KEY_RIGHT: direction = droite()
    elif c == curses.KEY_UP: direction = haut()
    elif c == curses.KEY_DOWN: direction = bas()

# remettre l'écran en "bon" état
curses.endwin()
print "\n\n\tF I N I !\n\n"

```

```

# pour que c existe à la ligne suivante
# tant qu'on n'a pas appuyé sur la touche 'q'
# effacer l'écran
# dessin du bord du terrain
# calcule la nouvelle position et place le serpent
# ne garde que la fin du serpent (taille invariable)
# un tour de plus
# si on passe 5 tours ...
# ... remise du compteur à zéro
# ... fait grandir le serpent
# si on touche le bord ...
# ... affichage du message "Perdu!"
# ... sortie de la boucle while (donc fin du programme)
# attente pour ralentir (arbitraire!)
# saisie de l'appui sur une touche du clavier
# on tourne à gauche
# on tourne à droite
# on tourne à gauche
# on tourne à droite
# sortie du mode "curses"
# message de fin!

```



## **Annexe A. Où trouver ce document ?**

Ce document est disponible sous plusieurs formats sur le web:

- Un seul document HTML : <http://www.grappa.univ-lille3.fr/polys/python2/python.html> (lourd à charger, mais facile à sauvegarder ou à imprimer)
- Plusieurs pages HTML : <http://www.grappa.univ-lille3.fr/polys/python2/index.html> (plus faciles à consulter)
- PDF : <http://www.grappa.univ-lille3.fr/polys/python2/python.pdf>

*Annexe A. Où trouver ce document ?*

## Annexe B. C'est quoi Python ?

**Note** : Ce texte est issu de <http://www.linux-center.org/articles/9812/python.html>.

Python est un langage portable, dynamique, extensible, gratuit, qui permet (sans l'imposer) une approche modulaire et orientée objet de la programmation. Python est développé depuis 1989 par Guido van Rossum et de nombreux contributeurs bénévoles.

Détaillons un peu les principales caractéristiques de Python, plus précisément, du langage et de ses deux implantations actuelles :

- Python est portable, non seulement sur les différentes variantes d'UNIX, mais aussi sur les OS propriétaires: MacOS, BeOS, NeXTStep, MS-DOS et les différentes variantes de Windows. Un nouveau compilateur, baptisé JPython, est écrit en Java et génère du bytecode Java.
- Python est gratuit, mais on peut l'utiliser sans restriction dans des projets commerciaux.
- Python convient aussi bien à des scripts d'une dizaine de lignes qu'à des projets complexes de plusieurs dizaines de milliers de lignes.
- La syntaxe de Python est très simple et, combinée à des types de données évolués (listes, dictionnaires...), conduit à des programmes à la fois très compacts et très lisibles. A fonctionnalités égales, un programme Python (abondamment commenté et présenté selon les canons standards) est souvent de 3 à 5 fois plus court qu'un programme C ou C++ (ou même Java) équivalent, ce qui représente en général un temps de développement de 5 à 10 fois plus court et une facilité de maintenance largement accrue.
- Python gère ses ressources (mémoire, descripteurs de fichiers...) sans intervention du programmeur, par un mécanisme de comptage de références (proche, mais différent, d'un *garbage collector*).
- Il n'y a pas de pointeurs explicites en Python.
- Python est (optionnellement) *multi-threadé*.
- Python est orienté-objet. Il supporte l'héritage multiple et la surcharge des opérateurs. Dans son modèle objets, et en reprenant la terminologie de C++, toutes les méthodes sont virtuelle.
- Python intègre, comme Java ou les versions récentes de C++, un système d'exceptions, qui permettent de simplifier considérablement la gestion des erreurs.
- Python est dynamique (l'interpréteur peut évaluer des chaînes de caractères représentant des expressions ou des instructions Python), orthogonal (un petit nombre de concepts suffit à engendrer des constructions très riches), reflectif (il supporte la métaprogrammation, par exemple la capacité pour un objet de se rajouter ou de s'enlever des attributs ou des méthodes, ou même de changer de classe en cours d'exécution) et introspectif (un grand nombre d'outils de développement, comme le debugger ou le profiler, sont implantés en Python lui-même).
- Comme Scheme ou SmallTalk, Python est dynamiquement typé. Tout objet manipulable par le programmeur possède un type bien défini à l'exécution, qui n'a pas besoin d'être déclaré à l'avance.
- Python possède actuellement deux implémentations. L'une, interprétée, dans laquelle les programmes Python sont compilés en instructions portables, puis exécutés par une machine virtuelle (comme pour Java, avec une différence importante: Java étant statiquement typé, il est beaucoup plus facile d'accélérer l'exécution d'un programme Java que d'un programme Python). L'autre, génère directement du bytecode Java.
- Python est extensible: comme Tcl ou Guile, on peut facilement l'interfacer avec des bibliothèques C existantes. On peut aussi s'en servir comme d'un langage d'extension pour des systèmes logiciels complexes.
- La bibliothèque standard de Python, et les paquetages contribués, donnent accès à une grande variété de services: chaînes de caractères et expressions régulières, services UNIX standard (fichiers, pipes, signaux, sockets, threads...), protocoles Internet (Web, News, FTP, CGI, HTML...), persistance et bases de données, interfaces graphiques.
- Python est un langage qui continue à évoluer, soutenu par une communauté d'utilisateurs enthousiastes et responsables, dont la plupart sont des supporters du logiciel libre. Parallèlement à l'interpréteur principal, écrit en C et maintenu par le créateur du langage, un deuxième interpréteur, écrit en Java, est en cours de développement.

Les domaines d'application naturels de Python incluent entre autres :

## Annexe B. C'est quoi Python ?

- L'apprentissage de la programmation objet.
- Les scripts d'administration système ou d'analyse de fichiers textuels.
- Tous les développements liés à l'Internet et en particulier au Web: scripts CGI, navigateurs Web, moteurs de recherche, agents intelligents, objets distribués...
- L'accès aux bases de données (relationnelles).
- La réalisation d'interfaces graphiques utilisateurs.
- Le calcul scientifique et l'imagerie. Python ne sert alors pas à écrire les algorithmes, mais à combiner et mettre en oeuvre rapidement des bibliothèques de calcul écrites en langage compilé (C, C++, Fortran, Ada...).
- Le prototypage rapide d'applications. L'idée générale est de commencer par écrire une application en Python, de la tester (ou de la faire tester par le client pour d'éventuelles modifications du cahier des charges). Trois cas peuvent alors se présenter :
  - Les performances sont satisfaisantes, après optimisation éventuelle du code Python. On livre alors le produit tel quel au client.
  - Les performances ne sont pas satisfaisantes, mais l'analyse de l'exécution du programme (à l'aide du profiler de Python) montre que l'essentiel du temps d'exécution se passe dans une petite partie du programme. Les fonctions, ou les types de données, correspondants sont alors réécrits en C ou en C++, sans modification du reste du programme.
  - Sinon, il est toujours possible de réécrire tout le programme, en utilisant la version Python comme un brouillon.

Même dans le pire des trois cas, il est très vraisemblable que le temps de développement aura été sensiblement plus court que si le programme avait été développé directement en C ou en C++.

Le site officiel de Python est <http://www.python.org>. On y trouvera la distribution officielle, de nombreux paquetages contributés, les compte-rendus des six conférences Python qui se sont déjà tenues à ce jour.

## Annexe C. D'autres sources d'information

### *Python HOWTO Documents*<sup>1</sup>

Les *HowTos* de Python ; c'est là que vous trouverez, entre autres, *Curses Programming with Python*<sup>2</sup> et *Python Advocacy HOWTO*<sup>3</sup> ci-dessous.

### *Curses Programming with Python*<sup>2</sup>

Clair, concis ; tout ce qu'il faut savoir pour utiliser `curses` avec Python.

### *Python Advocacy HOWTO*<sup>3</sup>

Pourquoi utiliser Python ?

### *X/Open Curses, Issue 4 Version 2 Reference Pages*<sup>4</sup>

La liste de toutes les procédures utilisables dans `curses`.

### *Curses, constants*<sup>5</sup>

### *Manipulating Characters with Curses*<sup>6</sup>

### *Curses, constants*<sup>7</sup>

Quelques pages qui dressent la liste des constantes utilisées dans `curses`.

- 
1. <http://www.amk.ca/python/howto/>
  2. <http://www.amk.ca/python/howto/curses/>
  3. <http://www.amk.ca/python/howto/advocacy/>
  2. <http://www.amk.ca/python/howto/curses/>
  3. <http://www.amk.ca/python/howto/advocacy/>
  4. <http://www.opengroup.org/onlinepubs/007908799/cursesix.html>
  5. <http://www.oopweb.com/Python/Documents/Official/Volume/lib/node183.html>
  6. [http://www.unet.univie.ac.at/aix/aixprgpd/genprogc/manip\\_charac\\_wcurses.htm#A55C21ab3](http://www.unet.univie.ac.at/aix/aixprgpd/genprogc/manip_charac_wcurses.htm#A55C21ab3)
  7. <http://python.active-venture.com/lib/node218.html>

