Travaux Pratiques
Initiation à la programmation avec Python 3
Feuille n.3
Boucles for, while et fonctions

Construire un programme **indépendant** (*shebang*, `chmod +x`) pour chaque exercice. Ceci n'empêche pas de tester des bouts de code dans l'interpréteur, avant de les intégrer dans le programme. Tous les résultats présentés par les programmes doivent être clairement compréhensibles à l'utilisateur.

1 Boucles `for` (simples ou imbriquées)

Définissez avant tout, et enregistrez dans un fichier, une fonction :

```
affiche_blancs_puis_etoiles(B,E)
```

qui affichera `B` caractères blancs (espaces), puis `E` caractères `"*"`, puis le retour à la ligne (c'est-à-dire `"\n"`, que `print` ajoute sans qu'on lui demande). Attention, cette fonction est tellement simple que cela peut être déroutant : c'est un simple appel à `print`. **Suggestion** : pour comprendre ce qui vous reste à faire, testez dans l'interpréteur Python 3 l'instruction suivante :

```
print("aaa","bbb",sep="")
```

Une fois écrite, vous pourrez copier-coller la définition pour pouvoir l'utiliser (l'appliquer) dans les exercices suivants. Plus généralement, essayez autant que possible de réutiliser le code écrit : si un exercice peut être résolu plus facilement en utilisant la solution d'un exercice précédent, transformez cette dernière en fonction et le tour est joué !

Attention : dans les exercices (b) il se peut (c'est même probable) qu'il soit nécessaire **d'imbriquer** les boucles les unes dans les autres...

1. (a) Écrire un programme `rectangle_etoiles.py` qui demande à l'utilisateur une *largeur* et une *hauteur* de rectangle, puis affiche le rectangle d'étoiles correspondant. Exemple d'exécution :

   ```
   Affichage d'un rectangle d'étoiles.
   Quelle hauteur ? 3
   Quelle largeur ? 7
   *******
   *******
   *******
   ```

   (b) Améliorer le programme en demandant à l'utilisateur s'il souhaite *transposer* le rectangle (la hauteur devient la largeur et réciproquement). Si l'utilisateur le souhaite, le rectangle sera alors dessiné dans l'autre sens. Par exemple, le nouveau programme donnera le résultat illustré ci-dessus lorsque l'utilisateur saisira comme hauteur 7, comme largeur 3, et demandera de transposer le rectangle.

2. (a) Écrire un programme `triangle_etoiles.py` qui demande à l'utilisateur une *hauteur,* puis affiche le triangle rectangle (isocèle) d'étoiles correspondant. Exemple d'exécution :

```
Affichage d'un triangle rectangle d'étoiles.
Quelle hauteur ? 5
*
**
***
****
*****
```

(b) Ré-écrire une autre version du programme qui dessine le triangle à l'envers (de la ligne avec le plus d'étoiles à celle avec une seule étoile).

3. (a) Écrire un programme `triangle_etoiles_droit.py` qui demande à l'utilisateur une *hauteur,* puis affiche le triangle rectangle (isocèle) d'étoiles correspondant, avec l'angle à droite. Exemple d'exécution :

```
Affichage d'un triangle rectangle d'étoiles (angle à droite).
Quelle hauteur ? 5
    *
   **
  ***
 ****
*****
```

(b) Ré-écrire une autre version du programme qui dessine le triangle à l'envers (de la ligne avec le plus d'étoiles à celle avec une seule étoile).

4. (a) Écrire un programme `triangle_isocele_etoiles.py` qui demande à l'utilisateur une *hauteur,* puis affiche le triangle isocèle d'étoiles correspondant. Exemple d'exécution :

```
Affichage d'un triangle isocèle d'étoiles.
Quelle hauteur ? 5
    *
   ***
  *****
 *******
*********
```

(b) Ré-écrire une autre version du programme qui dessine le triangle à l'envers (de la ligne avec le plus d'étoiles à celle avec une seule étoile).

5. (a) Écrire un programme `sapin_de_noel.py` qui demande à l'utilisateur une *hauteur* du sapin, puis affiche deux triangles isocèles, un premier de la hauteur donnée, le second (tronc) de hauteur divisée par deux. La première ligne du second triangle (la pointe du tronc) sera fusionnée avec la dernière ligne du premier triangle (le feuillage du sapin). Exemple d'exécution :

```
Affichage d'un sapin de Noël.
Quelle hauteur ? 6
      *
     ***
    *****
   *******
  *********
 ***********
     ***
    *****
```

2 Traitement des tableaux (listes) avec boucles `for`

On travaille avec les *tableaux*, c'est-à-dire les *listes* de Python. Chaque exercice correspond à une fonction à écrire et à tester (en l'appliquant plusieurs fois si possible). À noter que la **longueur** d'une liste en Python peut être obtenue par la fonction prédéfinie ("primitive") `len()`. N'utilisez pas les fonctions (ou méthodes) fournies par le langage : programmez votre propre version !

1. Calcul du *maximum* des éléments d'un tableau de nombres : `max(tab)` renvoie le plus grand nombre dans `tab`.
2. Calcul du *minimum* des éléments d'un tableau de nombres : `min(tab)` renvoie le plus petit nombre dans `tab`.
3. Calcul de la *moyenne* des éléments d'un tableau de nombres : `moyenne(tab)` renvoie le flottant moyenne.
4. Recherche d'un élément dans un tableau : `recherche(x, tab)` renvoie un résultat booléen exprimant la présence de `x` dans `tab`.
5. Tri des éléments dans un tableau : `est_ordonne(tab)` renvoie un résultat booléen exprimant le fait que le tableau `tab` soit trié du plus petit au plus grand (ordre croissant).
6. Unicité des éléments dans un tableau : `unicite(tab)` renvoie un résultat booléen exprimant l'absence de répétitions dans `tab`.
7. Inverser les éléments d'un tableau : `reverse(tab)` renvoie un tableau avec les mêmes éléments que `tab` mais se présentant dans l'ordre inverse (le 1er devient le dernier, le 2ème devient l'avant-dernier, ainsi de suite).

3 Boucles et fonctions (procédures) avec la tortue

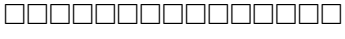
Rappel : dans l'interpréteur Python, l'instruction `from turtle import *` permet d'utiliser la tortue avec les fonctions suivantes :

`reset()`	on efface le tableau et on recommence
`position()`	pour connaître sa position actuelle
`goto(x,y)`	aller à la position de coordonnées (x,y)
`forward(x)`	avancer de x pixels
`backward(x)`	reculer de x pixels
`up()`	relever le crayon pour se déplacer sans laisser de traces
`down()`	abaisser le crayon pour tracer
`color(x)`	utiliser la couleur du crayon x
`left(x)`	tourner à gauche d'un angle x
`right(x)`	tourner à droite d'un angle x
`width(x)`	utiliser un tracé d'épaisseur x
`fill(x)`	remplir ou pas (x de type `bool`) un contour fermé
`write(x)`	écrire le texte x (de type `str`)

3.1 Exercices

Consigne : pour tous les exercices, il faut qu'à la fin de la procédure, une fois la figure tracée, la tortue revienne exactement dans la même position et orientation d'avant l'appel.

1. Définir et tester la procédure `trace_carre(L,C)`, qui dessine un carré de *longueur* `L` et de *couleur* `C`.
2. Définir et tester la procédure `trace_triangle(L,C)`, qui dessine un triangle équilatère de *longueur* `L` et de *couleur* `C`.
3. Définir et tester la procédure `trace_carres(N,L,C)`, qui dessine `N` carrés chacun de *longueur* `L` et *couleur* `C` les uns après les autres sur une même ligne imaginaire :



4. Définir et tester la procédure `trace_triangles(N,L,C)`, qui dessine `N` triangles équilatères chacun de *longueur* `L` et *couleur* `C` les uns après les autres sur une même ligne imaginaire :

△△△△△△△△△△△△△△△△△△△△

5. Définir et tester la procédure `trace_carres_et_triangles(N,L,C1,C2)`, qui dessine `N` carrés et `N` triangles équilatères **de façon alternée**, chacun de *longueur* `L` et *couleur* `C1` pour les carrés, `C2` pour les triangles, les uns après les autres sur une même ligne imaginaire :

□ △ □ △ □ △ □ △ □ △ □ △ □ △ □ △ □ △ □ △ □ △ □ △ □ △

6. Définir et tester la procédure `trace_carres_et_triangles_en_spirale(L,C1,C2)`, qui dessine de façon alternée des carrés et triangles équilatères comme la fonction précédente, mais en spirale au lieu que sur une même ligne.