Travaux Pratiques
Initiation à la programmation avec Python 3
Feuille n.2
Listes, Itérations (for), Fonctions

1 Utiliser les listes

Ouvrez un interpréteur Python dans un terminal, si possible `ipython`, sinon `python`, pour tester vos réponses.

1. définir une liste, stockée dans la variable `L`, contenant les chaînes de caractères correspondants aux jours de la semaine à partir de `"lundi"` (puis `"mardi"`, `"mercredi"`, ..).

   (a) Que répond l'interpréteur pour les expressions suivantes ?

   ```
   L[0]
   L[1]
   L[0:2]
   L[2:6]
   L[:3]
   L[3:]
   L[0:7:2]
   ```

   (b) Est-ce que l'**extraction d'éléments** pour les listes suit le même principe que pour les chaînes de caractères ? Pour répondre à cette question, définissez la chaîne :

   ```
   Z="ABCDEFG"
   ```

   et comparez les résultats obtenus avec `L` en faisant exactement les mêmes opérations sur `Z` :

   ```
   Z[0]
   Z[1]
   Z[0:2]
   Z[2:6]
   Z[:3]
   Z[3:]
   Z[0:7:2]
   ```

   (c) Est-ce que la **concaténation** (`+`) s'exprime de la même manière ?

   ```
   Z + Z
   L + L
   Z * 3
   L * 3
   ```

Remarque : plusieurs autres opérations sont disponibles sur le listes mais elles demandent une syntaxe orientée *programmation objet*, que nous n'avons pas encore introduit (ce sera le cas dans la suite du cours).

2. utiliser la fonction `range` pour générer les listes suivantes (tester en composant avec la fonction `list` comme vu en CM) :
   — la liste des entiers entre 0 et 9
   — la liste des entiers entre 0 et 99
   — la liste des entiers **pairs** entre 0 et 99
   — la liste des entiers **impairs** entre 0 et 99
   — la liste des entiers **multiples de 3** entre 0 et 99
   — la liste des entiers **multiples de 5** entre 0 et 99
   — la liste des entiers **multiples de 10** entre 0 et 99
   — la liste des entiers **multiples de 50** entre 0 et 1000

2 Utiliser les boucles

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. Écrire un programme qui affiche tous les jours de la semaine (réutiliser la liste `L` définie auparavant), un par ligne
2. Écrire un programme qui affiche tous les jours de la semaine avec l'index associé, ligne par ligne :

   ```
   lundi (jour 0)
   mardi (jour 1)
   ...
   dimanche (jour 6)
   ```

3. Modifier le programme précédent pour que la numérotation commence à 1 :

   ```
   lundi (jour 1)
   mardi (jour 2)
   ...
   dimanche (jour 7)
   ```

4. Modifier le programme précédent pour un affichage plus convivial du style (attention au cas particulier du lundi) :

   ```
   Le 1er jour de la semaine est lundi
   Le 2ème jour de la semaine est mardi
   ...
   Le 7ème jour de la semaine est dimanche
   ```

5. Écrire un programme faisant l'appel sur une liste d'étudiants, comme par exemple :

   ```
   ETUDIANTS = ["Bobo", "Coco", "Lolo", "Mimi", "Toto", "Zaza"]
   ```

   Pour chaque étudiant, le programme demandera à l'utilisateur si tel étudiant est présent ou pas en classe. Les réponses seront enregistrées dans une deuxième liste `PRESENT`, contenant des valeurs booléennes (`True` ou `False`). À la fin de l'appel, le programme affichera à l'utilisateur l'ensemble des étudiants présents, **puis** l'ensemble des étudiants absents.

6. En réutilisant les solutions de l'exercice 1.2, écrire différents programmes pour calculer :
   — la **somme** de tous les entiers **pairs** entre 0 et 99
   — la **somme** de tous les entiers **impairs** entre 0 et 99
   — le **produit** de tous les entiers **multiples de 11** entre 0 et 99

3 Utiliser des sous-programmes (fonctions)

Utilisez l'intépréteur Python pour ces premiers exercices (les fonctions tiennent sur une ligne). Attention : les fonctions **renvoient** des valeurs, ce qui n'a aucun rapport avec l'affichage (qui est plutôt une façon de "livrer" une valeur à l'utilisateur humain).

1. Définir et tester la fonction `succ`, qui renvoie le succésseur de son argument
2. Définir et tester la fonction `double`, qui renvoie son argument multiplié par 2
3. Définir et tester la fonction `triple`, qui renvoie son argument multiplié par 3
4. Définir et tester la fonction `surface_carre`, qui renvoie le carré de son argument
5. Définir et tester la fonction `surface_rectangle`, qui renvoie le produit des ses deux arguments

Construire un programme **indépendant** (*shebang*, `chmod +x`) pour le prochain exercice.

1. Restructurer le programme faisant l'appel (2.5) en définissant une fonction `repond_present(E)` qui affiche le nom de l'étudiant `E`, pose la question si l'étudiant est présent, et renvoie le booléen correspondant à la réponse (oui/non) de l'utilisateur. La partie principale du programme devra être maintenant une boucle `for` faisant appel au sous-programme `repond_present`.

4 Boucles et fonctions (procédures) avec la tortue

Rappel : dans l'interpréteur Python, l'instruction suivante :

```
from turtle import *
```

a l'effet d'apporter un certain nombre d'outils pour dessiner sur un écran graphique. Le trait correspond à la piste laissée derrière elle par une tortue virtuelle dont on peut contrôler les mouvements. Les fonctions principales sont :

`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`)

Avec l'interpréteur `ipython` vous pouvez avoir de l'aide sur toutes ces fonctions en utilisant le point d'interrogation juste après le nom de la fonction. Par exemple, il vous suffira de taper `color?` pour avoir de l'aide sur la fonction `color()`.

Remarque : Nous allons écrire des **procédures**, c'est-à-dire des fonctions qui ne renvoient pas de valeur. Elles ne renvoient pas de valeur mais ont un effet visible sur l'écran graphique de la tortue.

4.1 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`, qui dessine un carré de *longueur* définie par son unique argument.
2. Définir et tester la procédure `trace_rectangle`, qui dessine un rectangle de *longueur* et *hauteur* définies par ses deux arguments.
3. Définir et tester la procédure `trace_carres_emboites`, qui dessine plusieurs carrés les uns dans les autres qui partagent l'angle en bas à gauche. La procédure a trois paramètres : le premier définit la *longueur* du plus petit carré, le deuxième indique le *nombre* de carrés à dessiner, et le troisème paramètre définit l'*augmentation* (différence) de longueur entre deux carrés voisins. Suggestion : utiliser la fonction `trace_carre` du point 1, avec une boucle.
4. Définir et tester la procédure `trace_spirale`, qui dessine une spirale rectangulaire dont le trait initial est de *longueur* définie par un premier argument, et dont l'*augmentation* de la longueur du trait (lors des changements de direction) est donné par un second argument.