I) Fonctions

Exercice 1
Soit (u_n) la suite définie par $u_0 = -10$ et $u_{n+1} = \frac{3}{4}u_n + 5$.

1) Quelle est le bon type de données pour représenter les u_n ?

2) On peut montrer (mathématiquement) que (u_n) converge. Si on note ℓ sa limite, elle vérifie $\ell = \frac{3}{4}\ell + 5$ et donc ici $\ell = 20$. À l'aide d'un programme Python, calculer n tel que u_n soit à 10^{-3} de sa limite.

3) Écrire une fonction `rang` qui calcule le rang n à partir duquel u_n est à une distance ε de sa limite. On affichera le résultat pour $\varepsilon = 10^{-2}$, 10^{-5}, 10^{-50}. Expliquer ce qui se passe dans le dernier cas.

4) Adapter `rang` pour une suite $u_{n+1} = au_n + b$ (on imposera $|a| < 1$ pour que la suite converge), où a, b et u_0 sont des arguments de la fonction.

5) Adapter `rang` pour que la relation de récurrence $u_{n+1} = f(u_n)$ et la limite soit désormais des arguments de la fonction. Tester avec $f(x) = \sqrt{2+x}$, $u_0 = 0$, et $\ell = 2$.

Exercice 2 (bibliothèque `math`)

1) Importer les fonctions cosinus et sinus de la bibliothèque `math`.

2) Vérifier les valeurs de $\cos^2(x) + \sin^2(x)$ pour différente valeurs de x.

3) De même, conjecturer la valeur de $\dfrac{1-\cos(x)}{x^2}$ puis de $\dfrac{1}{x^4}\Big(1 - \dfrac{x^2}{2} - \cos(x)\Big)$ lorsque x tends vers 0.

Exercice 3 (bibliothèque `turtle`)

1) Importer toutes les fonctions de la bibliothèque `turtle` avec le préfixe `t` (parce que sinon c'est pénible).

2) Sachant que `mainloop()` est juste là pour que la fenêtre ne se ferme pas, que fait la suite de commandes :

```
t.down()
t.forward(40)
t.left(90)
t.forward(40)
t.up()
t.forward(40)
t.mainloop()
```

3) Faites un petit dessin de votre choix.

II) Listes

Définition 1 Une liste est une suite finie d'objets de n'importe quels types. Elle peut avoir une longueur arbitraire.

Exercice 4 (Premières manipulations)
Dans un interpreteur.

1) Entrer les listes suivantes : `L=[2,3.5,"vacances",True]` et `M=[-10,False,[4,5,6]]`

2) Afficher la longueur de la liste.

3) Que vaut l'élément `L[0]` ? `L[1]` ? Changer la valeur de l'élément d'indice `0` de `L`. Effacer l'élément d'indice `3` de `L`.

4) Accéder au `5` contenu dans `M`.

5) Ajouter un élément (de votre choix) au bout de la liste `L`.

6) Avec `range` : Définir la liste `N=[0,1,2,3,4,5,6]`.

7) Qu'affiche `N[2:5]` ?

8) Remplacer les éléments `2,3,4` par `4,6,8` dans `N`.

9) Intercaler les éléments `"Très","bonnes"` avant `"vacances"` dans `L`.

Exercice 5
Reprendre la fonction de l'exercice 1, dernière question. Stocker les termes de la suite dans une liste `L` retournée par la fonction.

Exercice 6 (Pointeurs)
Dans un interpreteur jusqu'à la fonction.

1) Effectuer les affectations suivantes : `L=[0,1,2,3,4,5,6]`, `M=L`, `Lcopie=L[:]`.

2) Effacer l'élément d'indice `0` de `L`. Que valent les trois listes ? Afficher les trois listes pour vérifier.

3) Dans un éditeur, rentrer le code suivant :

```
def f(L):
  L[0]=0
  return "pouet"

M=[1]
f(M)
print(M)
```

Constater le résultat.

4) Même programme en remplaçant `f(M)` par `f(M[:])`.

Pourquoi un tel comportement dans les fonctions ? Ce qui est passé à la fonction dans le premier cas, c'est l'adresse mémoire où se situe la liste. Dans la fonction, on appelle cette adresse `L`, mais elle pointe vers le contenu de `M`. Donc si on modifie l'intérieur de `L`, on est, en fait, entrain de modifier l'intérieur de `M` (c'est la même chose).
Lorsqu'on écrit `f(M[:])`, on passe en argument une *copie* de la liste `M`, donc `M` n'est pas modifiée. Le défaut, c'est l'encombrement : une liste peut être quelque chose d'énorme (par exemple contenir un million de flottants : $10^6 \times 64$ bit $= 10^6 \times 8$ octets $= 8$ Mo). On ne veut pas la dupliquer en mémoire à chaque appel de fonction !