I11 - Programmation I : Python
TP 6

Objectifs
- Implanter et utiliser des fonctions
- Utiliser des modules

Notions
- Passage de paramètres
- Visibilité, durée de vie

Consignes
- Les étudiants de la licence SI utiliseront un terminal de commande Unix pour toutes les manipulations du système de fichiers.
- Les étudiants des licences Mass/Maths/PC utiliseront le navigateur de fichiers.
- Les scripts doivent commencer par

```
# fichier: modele_script.py
# auteur: Nom prenom
```

---

1 Fonctions et structures de données

Exercice 1. Fonctions de traitement des listes.

Soit la liste `ll = [4, 7, 9, 3, 5, 4, 1]` définie en global, écrire les fonctions suivantes, les appeler avec les paramètres effectifs `ll` et 4 et donner leur résultat :

1. `liste_existe` qui prend deux paramètres, une liste et un élément à chercher dans la liste, et qui renvoie vrai ou faux en fonction de la présence ou non de l'élément dans la liste,
2. `liste_index` qui prend deux paramètres, une liste et un élément à chercher dans la liste, et qui retourne le premier index de l'élément dans la liste si celui-ci est présent et -1 sinon,
3. `liste_index_tous` qui prend deux paramètres, une liste et un élément à chercher dans la liste, et qui retourne tous les index référençant l'élément dans la liste si celui-ci est présent, une liste vide sinon.

Exercice 2. Fonction utilisant une fonction.

Sachant qu'une année bissextile est divisible par 4 (mais non par 100) ou par 400 et que le mois de février compte alors 29 jours, écrire la fonction qui prend un entier correspondant à une année en paramètre et qui retourne vrai ou faux si celle-ci est ou non bissextile.

En utilisant cette première fonction, écrire une deuxième fonction qui retourne le nombre de jours à partir de deux paramètres : le nom du mois et l'entier correspondant à l'année. On utilisera les listes suivantes : `lnj = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]` et `lnm = ['janvier', 'février', 'mars', 'avril', 'mai', 'juin', 'juillet', 'août', 'septembre', 'octobre', 'novembre', 'décembre']`.

Afficher le résultat de l'appel à cette deuxième fonction puis la réécrire en utilisant le dictionnaire `dnbjm = {'janvier': 31, 'février': 28, ...}`.

Exercice 3. Calculatrice en polonaise inverse

En adaptant éventuellement la fonction `monExpression()` étudiée durant le TD 6, simuler une calculatrice fonctionnant en notation polonaise inverse.

(source wiki) La notation polonaise inverse, également connue sous le nom de notation post-fixée, permet d'écrire de façon non ambigüe les formules arithmétiques sans utiliser de parenthèses. Dérivée de la notation polonaise présentée en 1920 par le mathématicien polonais Jan Łukasiewicz, elle s'en différencie par l'ordre des termes, les opérandes y étant présentés avant les opérateurs et non l'inverse. Par exemple, l'expression « $3 \times (4 + 7)$ » peut s'écrire en NPI sous la forme « $4\ 7 + 3\ \times$ », ou encore sous la forme « $3\ 4\ 7 + \times$ ».

La réalisation de calculatrices NPI est basée sur l'utilisation d'une pile, c'est-à-dire, que les opérandes sont ajoutés au fur et à mesure en haut de la pile, et les résultats des calculs sont eux aussi retournés en haut de la pile. Dans le cadre de cet exercice, la pile sera simulée par une liste (de 10 éléments maximum) dont la gestion de l'index courant permettra de savoir si la pile est vide, si elle est pleine, d'ajouter un opérande et de le récupérer (ce qui l'enlève de la liste). Ecrire les fonctions suivantes :

- `pile_vide()` qui retourne un booléen,
- `pile_pleine()` qui retourne un booléen,
- `pile_empiler(pop)` qui ajoute un opérande dans la liste à l'index courant et augmente l'index de 1 si la pile n'est pas pleine,
- `pile_depiler()` qui diminue l'index de 1 et retourne l'opérande si la pile n'est pas vide,

ainsi que le programme principal qui demandera les opérandes et les opérateurs jusqu'à la saisie d'un caractère de fin (par exemple '.') puis affichera le résultat.

On rappelle que la fonction `monExpression()` reconstruit une expression arithmétique valide à partir de ces paramètres puis retourne l'expression arithmétique ainsi que le résultat de son évaluation (on utilisera pour cela la fonction interne `eval()` qui fait appel à l'interprète Python pour évaluer une chaîne de caractères et retourner son résultat).

2 Modules

Dans les exercices suivants, on fera appel aux modules `random` et `cng`. Ce dernier propose quelques fonctions qui simplifient le travail du programmeur qui souhaite avoir une sortie graphique 2D. Il comporte notamment les fonctions suivantes :

- `init_window(pnom, pla, pha)` : à partir du moment où l'on souhaite afficher une fenêtre graphique, cette fonction doit être la *première* appelée des fonctions du module `cng`,
- `main_loop()` : cette fonction doit être la *dernière* appelée des fonctions du module `cng` (c'est la boucle d'attente active propre aux GUI),
- `current_color(*args)` : définit la couleur courante, accepte soit un triplet RVB d'entiers compris entre 0 et 255, soit un nom (en anglais) de couleur,
- des fonctions de dessin : `point(px, py)`, `box(px1, py1, px2, py2)`, `circle(px, py, pr, pep=1)`, ...
- `refresh()` : permet de « forcer » l'affichage du dessin (si des instructions de dessin sont dans une boucle, l'affichage ne se fait par défaut qu'après la boucle).

Exercice 4. Test d'équirépartition

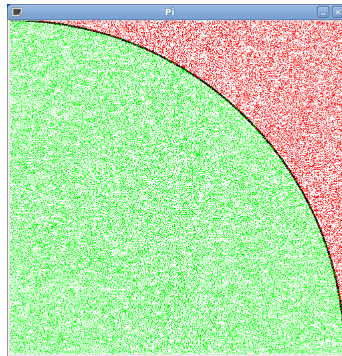
On souhaite vérifier le fonctionnement de l'un des générateurs de nombres aléatoires proposé par Python dans son module `random`. En effet, l'ordinateur étant par essence un automate déterministe, il n'est pas facile de créer un programme qui soit non prévisible i.e qui simule l'effet du hasard.

Ecrire un programme destiné à vérifier le fonctionnement du générateur de nombres aléatoires `randrange` de Python en simulant 10000 lancers d'un dé à 6 faces, en mémorisant les résultats dans une liste de 6 éléments, puis en affichant le nombre de sorties de chaque face.

Appliquer la même procédure en simulant 10000 lancers de deux dés à 6 faces puis présenter le résultat sous la forme d'un histogramme grâce au module `cng`. Quelle forme générale doit avoir cet histogramme si le tirage des nombres est bien équiréparti ?

EXERCICE 5. Évaluation du nombre π par la méthode Monte Carlo.

Soit un quart de cercle de rayon $R = 1$ inscrit dans un carré de coté 1.



L'aire du carré vaut $(R)^2$ soit 1. L'aire du quart de cercle vaut $\pi R^2/4$ soit $\pi/4$.

En choisissant N points aléatoires (à l'aide d'une distribution uniforme) à l'intérieur du carré, la probabilité que ces points se trouvent aussi dans le cercle est

$$p = \frac{\text{aire du cercle}}{\text{aire du carré}} = \frac{\pi}{4}$$

Soit n, le nombre points effectivement dans le cercle, il vient alors

$$p = \frac{n}{N} = \frac{\pi}{4},$$

d'où

$$\pi = 4 \times \frac{n}{N}.$$

1. Déterminer une approximation de π par cette méthode. Pour cela, pour N itérations, choisir aléatoirement les coordonnées d'un point entre 0 et 1 (fonction `random()` du module `random`), calculer la distance entre le centre du cercle et ce point et déterminer si cette distance est inférieure au rayon du cercle. Le cas échéant, le compteur n sera incrémenté.

   Que vaut l'approximation de π pour 1000 itérations ? 10000 ? 100000 ? Remarque : il faut compter environ 1 mn d'attente pour 1 million de points.

2. En utilisant le module `cng`, illustrer cette méthode en colorant en vert les points dans le cercle et en rouge les points en dehors du cercle. Afficher des résultats intermédiaires de l'approximation de π durant l'animation.

EXERCICE 6. Triangle de Pascal

En mathématiques, le triangle de Pascal, est une présentation des coefficients binomiaux dans un triangle.

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1

Premières lignes du triangle de Pascal.

(source wiki) Un algorithme, en langage formel, de construction du triangle de Pascal peut se présenter comme suit, en utilisant la relation de récurrence entre coefficients binomiaux :

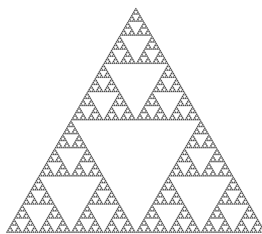
```
Algorithme Triangle de Pascal()

variables

    Entiers i, j, n, X
    Tableau de 1 à X de tableau de 1 à X d'entiers c (tableau bidimensionnel)

n ← 10        ▷ n est inférieur ou égal à la taille X utilisée dans le tableau c
c[0][0] ← 1
pour i de 1 a n faire
    c[i][0] ← 1
    pour j de 1 a i − 1 faire
        c[i][j] ← c[i-1][j-1] + c[i-1][j]
    finpour
    c[i][i] ← 1
finpour
```

1. Ecrire la fonction qui construit un triangle de Pascal pour un X donné.
2. En grisant les cases où apparaît un nombre impair et blanchissant les cases où apparaît un nombre pair, on obtient une image analogue au triangle de Sierpiński.



En utilisant le module `cng`, afficher un triangle de Sierpiński correspondant à la dimension du triangle de Pascal choisie.

Exercice 7. Dictionnaire

Soit la séquence nucléotidique suivante :

`ACCTAGCCATGTAGAATCGCCTAGGCTTTAGCTAGCTCTAGCTAGCTG`

En utilisant un dictionnaire, faites un programme qui répertorie tous les mots de 2 lettres qui existent dans la sequence (AA, AC, AG, AT, etc.) ainsi que leur nombre d'occurences puis qui les affiche à l'écran. Remarque : on rappelle que l'opérateur `in` permet de savoir si une clé est déjà présente dans un dictionnaire ou pas.