TP5 : Compléments de programmation

Exercice 1 *Avant de commencer (pour rappel)*

Démarrer le logiciel Spyder puis, dans le menu ≪ Fichier ≫ créer un nouveau fichier puis le sauvegarder avec un nom sous la forme TP5_votrenom. Dans la suite, on sauvegardera régulièrement son travail.

I Comparaison structure itérative et récursive

Exercice 2 *Coefficients binomiaux via la factorielle*

1. Écrire la fonction `facto(n)` qui renvoie $n!$.
2. On rappelle la formule pour $n \in \mathbb{N}$ et $k \in [\![0, n]\!]$:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

   Créer une fonction Python `binome_facto(n,k)` qui renvoie $\binom{n}{k}$ pour $k \in [\![0, n]\!]$ et 0 sinon. On pourra utiliser la fonction `facto`.
3. Modifier la fonction précédente afin qu'elle teste d'abord si les arguments sont entiers. Sinon, elle renverra un message du type `'un argument n est pas un entier'`. On utilisera `try` et `except`.
4. Créer une fonction Python `somme_coeff_binomiaux(n)` qui retourne la somme

$$\sum_{k=0}^{n} \binom{n}{k}.$$

   Tester la fonction et expliquer.

Exercice 3 *Coefficients binomiaux via la formule de Pascal*

1. Que vaut $\binom{n}{0}$ pour tout entier n ?
2. On rappelle la formule pour $n \in \mathbb{N}$ et $k \in [\![0, n]\!]$:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}.$$

   Créer une fonction Python `binome_recur(n,k)` qui de façon récursive la coefficient binomiale (on pourra comparer avec l'exercice précédent le résultat).

Exercice 4 *Évaluation du temps de calcul*

1. Importer le package `time` avec la commande `from time import *`.
2. Pour déterminer le temps de calcul nécessaire à l'exécution d'une fonction, par exemple la fonction `facto`, on peut écrire

```
def temps_facto(n):
    T1 = time()
    P=1
    for i in range(1,n+1):
        P = P*i
    T2 = time()
    return T2 - T1
```

   À noter qu'ici, on ne retourne pas le résultat de la fonction mais seulement le temps de calcul. Représenter le temps de calcul sur un graphique pour la fonction `facto` pour $n = 0, \ldots, 1000$.
3. Représenter le temps de calcul sur un même graphique pour les fonctions `binome_facto(n,int(float(n)/2))` et `binome_recur(n,int(float(n)/2))` pour $n = 0 \ldots, 200$. Interpréter.

II Une approximation de $\sqrt{2}$

Exercice 5 *Méthode de Héron*

On considère la suite (u_n) définie par récurrence par $u_0 = 2$ puis pour tout $n \in \mathbb{N}$:

$$u_{n+1} = \frac{1}{2}u_n + \frac{1}{u_n}.$$

1. Écrire une fonction Python `heron_iter(n)` calculant le n-ième terme de la suite de façon itérative.
2. Écrire une fonction Python `heron_recur(n)` calculant le n-ième terme de la suite de façon récursive.
3. Écrire une fonction Python `graph(n)` qui représente graphiquement n termes de la suite (u_n). Tester pour de petites valeurs de n.

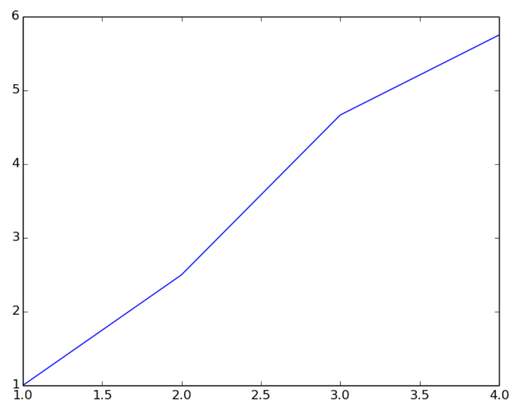
III Listes aléatoires et nombres d'occurrences

Exercice 6 *Liste aléatoire*

1. Importer le module `random` en entier.
2. Chercher dans l'aide l'utilité de la fonction `randint` avec `help(randint)`.
3. Avec la question précédente, créer une fonction Python `liste_alea(n,m)` qui renvoie une liste de taille n d'entiers naturels inférieurs ou égaux à m.

Exercice 7 *Jeu de pile ou face et loi des grands nombres*

1. Créer une fonction `pileouface(n)` qui crée une liste de taille n composée aléatoirement de 0 et de 1.
2. Calculer `(float(1)/n)*sum(pileouface(n))` pour $n = 10, 100, 1000, 10000$. Expliquer.
3. Créer une liste `L` avec 20 nombres et tester dans la console `L[:4]` et `L[:10]`. Expliquer l'intérêt de ces commandes.
4. Créer une fonction Python `repres(L)` qui, à partir d'une liste `L` représente, sur un graphique, pour k allant de 1 à la longueur de la liste, la moyenne des k premiers éléments. Par exemple, si $L = [1, 4, 9, 9]$, on obtiendra graphiquement les termes 1, 5/2, 14/3, 23/4 c'est-à-dire le graphique



5. Tester `repres(pileouface(1000))` et expliquer.

Exercice 8 *Théorème de la limite centrale*

1. Écrire une fonction Python `TCL(n,N)` qui renvoie qui effectue N tirages de `pileouface(n)` et renvoie une liste `L` de taille $n + 1$ qui contient dans `L[k]` le nombre de fois que `pileouface(n)` contenait k fois le nombre 1. Par exemple, en exécutant `TCL(4,5)`, on a obtenu les tirages : `[1,0,0,1]`, `[1,0,1,1]`, `[1,1,0,1]`,`[0,0,0,0]` et `[1,0,1,1]`, `TCL(4,5)` retournera `[1,0,1,3,0]`.
2. En ayant chargé le module `matplotlib.pyplot`, exécuter dans la console

```
>>> plot(range(1000),TCL(n,1000))
```

3. Interpréter ce résultat.

IV Compléments

Exercice 9 *Représentation des polynômes*

En Python, on peut représenter un polynôme par une liste. Par exemple : `[1,2,4]` représente le polynôme $1 + 2X + 4X^2$.

1. Que représente la liste `[1]*8` comme polynôme ?
2. Écrire une fonction `deg(L)` qui pour une liste `L` renvoie le degré du polynôme représenté.
3. Écrire une fonction `evaluation(L,x)` qui pour une liste `L` renvoie le polynôme représenté évalué en `x`. Par exemple `evaluation([1,2,4],1)` renvoie `7.0`.
4. Écrire une fonction `ajout_zero(L1,L2)` qui renvoie une liste de deux listes ayant la même longueur en complétant `L1` ou `L2` (la plus courte) par des 0.
5. Écrire une fonction `somme(L1,L2)` qui renvoie une liste représentant la somme des polynômes représentés par `L1` et `L2`.
6. Écrire une fonction `somme(L1,L2)` qui renvoie une liste représentant le produit des polynômes représentés par `L1` et `L2`.

Exercice 10 *Nombres d'occurrences dans une liste*

1. Écrire une fonction Python `presence(k,L)` qui retourne `True` si k est dans la liste L et `False` sinon.
2. Écrire une fonction Python `positions(k,L)` qui pour une liste d'entiers renvoie les positions de k dans cette liste. De façon plus précise, cette fonction doit retourner :

```
>>> positions(42, [12, 17, 42, 5])
[2]
>>> positions(24, [12, 17, 42, 5])
[ ]
>>> positions(42, [42, 12, 17, 42, 5])
[0, 3]
```

3. Écrire une fonction Python `nb_apparitions(L,m)` qui à partir d'une liste `L` crée une liste de `L_occ` de longueur $m + 1$ qui donne à l'indice i le nombre de fois qu'apparaît i dans `L`. De façon plus précise, cette fonction doit retourner :

```
>>> positions([1,4,4,5,6,7,7,0,1],7)
[1,2,0,0,2,1,1,2]
```

Exercice 11 *Flocon de Von Koch*

On introduit d'abord un outil mathématique. On appelle similitudes directe du plan complexe les composées d'homothéties, de rotations et de translations. Si M est un point du plan d'affixe z, son image par une similitude a pour affixe

$$f(z) = az + b$$

où $a, b \in \mathbb{C}$ avec $a \neq 0$ caractérisent la similitude de la façon suivante :

1. si $a = 1$, la fonction f représente la translation de vecteur d'affixe b ;
2. si $a \notin \{0, 1\}$, en posant $a = \rho e^{i\theta}$ et en notant ω l'unique complexe qui vérifie $f(\omega) = \omega$), la similitude représentée par f est la composée d'une rotation de centre Ω d'affixe ω et d'angle θ et d'une homothétie de rapport ρ.

On note :

- f_1 la similitude de centre O de coordonnées $(0, 0)$, de rapport $\frac{1}{3}$ et d'angle 0 ;
- f_2 la similitude qui transforme O en B de coordonnées $\left(\frac{1}{3}, 0\right)$ et A de coordonnées $(1, 0)$ en C de coordonnées $\left(\frac{1}{2}, \frac{1}{2\sqrt{3}}\right)$;
- f_3 la similitude qui transforme O en C et A en D de coordonnées $\left(\frac{2}{3}, 0\right)$;
- f_1 la similitude de centre A de coordonnées $(0, 0)$, de rapport $\frac{1}{3}$ et d'angle 0.

1. Calculer les fonctions f_1, f_2, f_3 et f_4.

2. Charger le module `cmath` et définir les fonctions précédentes en Python.
3. Définir une fonction `successeur(L)` où, pour une liste de nombre complexe `L = [a1, ...,an]` renvoie `[f_1(a1), ...,f_1(an),f_2(a1), ...,f_2(an),f_3(a1), ...,f_3(an),f_4(a1), ...,f_4(an)]`.
4. Créer une fonction Python `iteration(n)` qui, à partir de la liste `[0,1]` effectue n fois l'étape précédente.
5. Écrire une fonction Python `fractale(n)` qui représente une ligne polygonale qui joint les points de la liste précédente.
6. Écrire finalement une fonction Python `flocon(n)` qui permet d'obtenir la figure ci-contre :