Suites récurrentes en mathématiques

Exercice 1. 1. Écrire une fonction `Factorielle`, qui à $n \in \mathbb{N}$, renvoie $n!$ à l'aide d'une boucle.

2. Écrire une fonction `Factorielle`, qui à $n \in \mathbb{N}$, renvoie $n!$ en utilisant la récursivité.

Exercice 2. Soit $u_0 = 5$ on pose pour tout $n \in \mathbb{N}$, $u_{n+1} = 1 - \dfrac{1}{2+u_n}$. Procéder de même qu'à l'exercice précédent. Tester, vos programme avec $n = 100$, les fonctions doivent renvoyer vaut 0.6180339887498949

Exercice 3. 1. Rappeler la formule du triangle de Pascal.

2. Grâce à cette formule, écrire une fonction **récursive** `CoeffBinomial(n,p)` qui renvoie $\binom{n}{p}$. Pour l'initialisation, traiter à part les cas $\binom{n}{0}$ et $\binom{n}{n}$.
3. Calculer à la main $\binom{10}{3}$ et comparer avec la fonction `CoeffBinomial`.
4. Cela est-il satisfaisant pour `CoeffBinomial(28,11)` ?

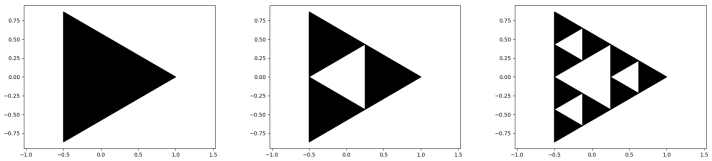
Exercice 4. Dans cet exercice, on n'utilisera pas la puissance `**` de Python.

1. Écrire une fonction `Puissance(x,n)` qui renvoie x^n.
2. Combien de multiplications sont faites pour calculer x^{32} ?
3. En remarquant que si $n = 2p$ est pair $x^n = (x^p) \times (x^p)$ et si $n = 2p+1$, $x^n = (x^p) \times (x^p) \times x$ écrire une fonction récursive qui renvoie x^n, le but étant de faire le moins de multiplications possibles.
4. Avec cette fonction, combien de multiplications pour calculer x^{32} ?

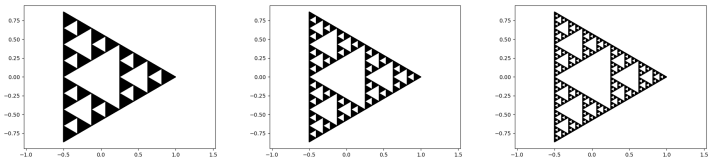
Récursivité en Python

Exercice 5. Créer une fonction récursive `Maximum(L)` qui va renvoyer le maximum d'une liste `L` (non vide). Pour cela, traiter le cas à part où `L` a un élément. Puis sinon, créer une liste `M` qui est la liste `L` sans son dernier élément, ensuite, calculer le maximum de `M` puis comparer ce maximum au dernier élément retiré.

Exercice 6. Le but de cet exercice est de dessiner une fractale appelée fractale de Sierpinski.



(a) À l'étape 0 (b) À l'étape 1 (c) À l'étape 2



(d) À l'étape 3 (e) À l'étape 4 (f) À l'étape 5

1. Créer une fonction `Triangle(A,B,C)`, qui ne renvoie rien, mais remplit en noir, le triangle des points A, B et C (chaque point étant défini comme une liste de la forme `[abscisse,ordonnée]` ou comme un couple `(abscisse,ordonnée)`), pour cela, on peut utiliser la fonction `plt.fill` qui fonctionne comme `plt.plot` : il faut une liste d'abcisses et une liste d'ordonnées (de même longueur), pour la couleur, on peut utiliser `color="black"`.

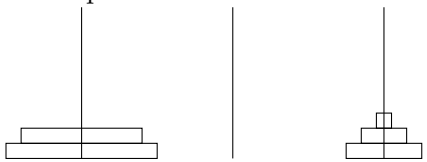
Grâce à cette fonction, on peut afficher le triangle à l'étape 0

```
A=[1,0]#A, B et C sont les trois points d'un triangle équilatéral
B=[-0.5,3**(0.5)/2]#Notez que les affixes de ces points sont les
C=[-0.5,-3**(0.5)/2]#racines cubiques de 1 : 1,j,j*j (voir exo TD5)
Triangle(A,B,C)
plt.show()
```

2. Écrire une fonction `Sierpinski(A,B,C,n)` **récursive** qui affiche le triangle de Sierpenski à l'étape n dans un triangle `ABC`. Pour cela, étant donné le triangle `ABC`, créer les milieux des côtés de ce triangle notés `I`, `J` et `K`. Remarquez qu'il faut que le petit triangle `IJK` soit blanc. De plus, il y a trois autres petits triangles, ces trois triangles doivent être coloriés grâce à la fonction Sierpinski à l'étape $n-1$.
3. Pour plus de fantaisie, modifier votre code pour que la couleur des triangles soient aléatoires.

Exercice 7. La tour de Hanoï est un célèbre casse-tête inventé par Édouard Lucas [1], il faut déplacer des disques de diamètres différents d'une tour de départ à une tour d'arrivé en utilisant une tour intermédiaire avec deux contraintes :

- On déplace un seul disque à la fois.
- On ne placer un disque que sur un disque plus grand ou sur un emplacement vide.



1. Chaque tour sera modélisée par une liste, ainsi, on a les listes `G`, `M` et `D`. Chaque disque sera modélisé par un entier n qui appartiendra à la liste, les disques les plus gros seront en premiers dans les listes. Au départ, tous les disques sont dans la tour gauche, ainsi `G=[n,n-1,n-2,...,1]`, tandis que `M` et `D` sont vides, initialiser `G`, `M` et `D`.
2. Écrire une fonction `DéplacerDisque(Liste1,Liste2)` qui prend le disque tout en haut de la tour représentée par `Liste1` l'enlève et le met tout en haut de la tour représentée par `Liste2`, on vérifiera si ce disque est bien plus petit que le dernier élément de `Liste2` (ou si cette liste est vide) (et on affichera une erreur sinon), cette fonction ne renverra rien, car elle modifie les listes par effet de bord.
3. Écrire une fonction récursive `Hanoi(G,M,D,n)` dont le but est de déplacer les n plus petits disques de `G` pour les placer dans `D`, pour cela réfléchir au cas $n = 1$, puis pour les autres cas, déplacer les $n-1$ plus petits disques de G vers M, déplacer le disque n de G vers D, puis placer les $n-1$ disques de M vers D.
4. Modifier la fonction `DéplacerDisque` pour qu'elle affiche les trois listes à chaque appel de cette fonction.
5. Déterminer C_n le nombre de déplacement à faire pour déplacer les n disques.

Exercice 8. Regardez le fichier `TP10Flocon` qui est sur CDP et programmez cette courbe de façon **récursive**. La création du gif est en bonus. Pour cela :

1. Professeur à Saint-Louis

- La commande `plt.savefig("MonNom.png")` enregistre l'image sous le nom `MonNom`,
- Utiliser la bibliothèque `Pil` grâce à `from PIL import Image`
- Créer la liste de toutes les images avec cette commande : `frames = [Image.open("nom image dépendant de i") for i in range(...)]`
- Enfin, convertir en gif : `frames[0].save('Flocon.gif', format = 'GIF', append_images = frames[1:],save_all = True,duration=300, loop=0)`