Informatique, composition n° 2, une correction

Exercice 1

1. Une fonction `factorielle(n)` qui renvoie la factorielle d'un entier $n \in \mathbb{N}$ peut-être :

```
def factorielle(n):
        if n==0 :
                return 1
        else :
                return n*factorielle(n-1)
```

2. On considère la fonction Python suivante `binom(n, p)` qui renvoie le cœfficient binomial $\binom{n}{p}$:

```
def binom(n, p):
        if not (0 <= p <= n):
                return 0
        return factorielle(n) // (factorielle(p) * factorielle(n - p))
```

• Lorsque l'on exécute `binom(30, 10)`, il y a $29+9+19+1$ multiplications, ce qui fait beaucoup.

• En notant que $\binom{30}{10} = \dfrac{30 \times 29 \times \cdots \times 21}{1 \times 2 \times \cdots \times 10}$ on voit qu'il est possible de réduire à 20 multiplications (et une division).

• Si on remplace la dernière ligne de la fonction `binom` par `return factorielle(n) / (factorielle(p) * factorielle(n - p))`, on obtient un résultat de type `float`. La commande `//` permet d'obtenir un résultat considéré de type *entier*.

3. • On voit de suite que :

$$\binom{n}{p} = \frac{n}{p}\binom{n-1}{p-1}.$$

.

• Une fonction récursive `binom_rec(n, p)` qui renvoie le cœfficient binomial $\binom{n}{p}$ peut-être :

```
def binom_rec(n, p):
        if 2*p>n :
                p=n-p
        if p==0:
                return 1
        else :
                return n*binom_rec(n, p)//p
```

4. • Il est facile de voir que pour $n \geqslant p \geqslant 1$, on a :

$$\binom{n}{p} = \binom{n-1}{p-1} + \binom{n-1}{p}$$

• Une fonction récursive `binom_rec2(n, p)` qui renvoie le cœfficient binomial $\binom{n}{p}$ peut-être :

```
def binom_rec2(n, p):
        if 2*p>n :
                p=n-p
        if p==0:
                return 1
        else :
                return binom_rec2(n-1, p-1)+binom_rec2(n-1, p)
```

5. Le code suivant remplit le triangle de Pascal par méthode ascendante en utilisant la formule de Pascal. Il commence par les côté gauche et droit du triangle (boucle `for k in range(2,n+1)`).

```
dico={}
        #Création d'un dictionnaire vide
def binom-asc(p,n):
        if 2*p>n :
                p=n-p
                        #On choisit le plus petit de p et n-p
        dico[(0,1)]=1
                #Affecte la valeur 1 à la clé (0,1) du dictionnaire
        dico[(1,1)]=1
                #Affecte la valeur 1 à la clé (1,1) du dictionnaire
        for k in range(2,n+1):
                dico[(0,k)]=1
                dico[(k,k)]=1
                #Affecte la valeur 1 à toutes les clés de la forme (0,k) ou (k,k)
                for i in range(1,k):
                        dico[(i,k)]=dico[(i,k-1)]+dico[(i-1,k-1)]
                                #Utilise la formule de Pascal pour remplir
                                #les valeurs des clés (i,k) du dictionnaire
        return dico[(p,n)]
                        #Renvoie la valeur de la clé (p,n) du dictionnaire
```

6. Une fonction non récursive `bernoulli(n)` qui renvoie une valeur approchée du nombre rationnel b_n peut-être :

```
def bernoulli(n):
        b = [1]
                        # b[k] vaut b_k
        for p in range(1, n + 1):
                b.append(-sum(binomial(p + 1, k) * b[k] for k in range(p)) / (p + 1))
                        # On calcule b_p et on l'ajoute \'a sa place, au bout de b.
        return b[-1]
```

7. La principe de mémoïsation consiste à stocker les valeurs calculées lors des différents appels récursifs. Un script peut-être en utilisant un dictionnaire :

```
dico={}
        #on va stocker les valeurs calculées des combinaisons
        #Les clés du dictionnaire sont de la forme (k,l) et la valeur associée sera k


def binom_rec3(n,p):
        if 2*p>n :
                p=n-p
        if (p,n) not in dico :
                if p==0 :
```

```
                        coeff=1
                else :
                        coeff=binom_rec3(p-1,n-1)+binom_rec3(p,n-1)
                dico[(p,n)]=coeff
        return dico[(p,n)]
```

Exercice 2

1. Une fonction Python `Maxliste`, qui mange une liste de nombres `L` et qui renvoie une liste de deux éléments dont le premier est le maximum de la liste et le second un indice où ce maximum est atteint peut-être :

```
def Maxliste(L):
        max=L[0]
        for i in range(1,len(L)):
                if L[i]>max:
                        max=L[i]
                        place=i
        return [max,place]
```

2. (a) La fonction donner va créer une liste `Occurence(L)` comportant 1001 nombres telle qu'à la place `v`, pour `v` entre 0 et 1000, la liste `Occurence(L)` contient le nombre de fois où la valeur `v` se trouve dans la liste `L`.

   (b) Une fonction `TriOcc` qui permet de trier par ordre croissant une liste `L` de nombres entiers compris (au sens large) entre 0 et 1000 qui utilise la fonction `Occurence` peut-être :

```
def TriOcc(L):
        N=[]
        M=Occurence(L)
        for i in range(len(M)):
                if M[i]!=0 :
                        for j in range(M[i]) :
                                N.append(i)
        return N
```

3. La fonction suivante où `L` est une liste de nombres entiers naturels implémente le tri par insertion.

```
def function(L):
        for i in range(1,len(L)):
                insertion=L[i]
                        #Nombre à insérer parmi une liste déjà triée
                j=i
                while j>0 and L[j-1]>insertion :
                        L[j]=L[j-1]
                        j=j-1
                        L[j]=insertion
                        #On trouve ici la place d'insertion de L[i] et on décale
        return L
```

4. Une fonction Python `fusion` qui mange deux listes d'entiers naturels `L1` et `L2` triées par ordre croissant et qui renvoie une liste `L` composées de tous les nombres des listes `L1` et `L2` triés par ordre croissant peut-être :

```
def fusion(L1,L2):
        L=[]
        i=0
                #compteur pour les éléments de L1
        j=0
                #compteur pour les éléments de L2
```

```
        n=len(L1)
        m=len(L2)
        for k in range(n+m):
                if i<n and (j==m or L1[i]<=L2[j]):
                        L.append(L1[i])
                        i=i+1
                                #On compare l'élément L1[i] à l'élément L2[j],
                                #pour déterminer lequel on rajoute à la liste L
                else :
                        L.append(L2[j])
                        j=j+1
        return L
```