DS 1 d'Informatique - IPT

Exercice 1

On donne le listing suivant :

```
1  def factorielle(n):
       f=1
3      for k in range(1,n+1):
           f=f*k
5      return f

7  def bin(k,n):
       # appelle la fonction factorielle préalablement créée
9      c=factorielle(n)/(factorielle(k)*factorielle(n-k))
       c=int(c) # corrige le type de c qui était en "float"
11     return c

13 def pasc(N):
       print('Les ',N,' premières lignes du triangle de Pascal :')
15     for n in range(N+1):
           print ('Ligne ',n,' : ',end='') # end='' c'est pour qu'il
       n'y ait pas de retour à ligne
17         for k in range(n+1):
               print(bin(k,n),' ',end='')
19         print() # retour à la ligne
```

1. Fonction `factorielle`

a. Proposer une ligne à ajouter à la fonction `factorielle` pour afficher le résultat avec une phrase explicite pour l'utilisateur.
Entre les lignes 4 et 5 avec la même indentation que la ligne 5 :

```
print(n,'! =',f,'  (la factorielle de ',n,' est ',f,')')
```

b. Décrire le fonctionnement de la fonction `factorielle`. *(entrée, traitement, sortie...)*

La fonction `factorielle` prend en entrée un entier naturel n.
Elle initialise la variable `f` à 1.
Elle réalise une boucle séquentielle avec l'entier `k` qui prend les valeurs successives de 1 jusqu'à n inclus.
Le nombre `f` est multiplié par chacune des valeurs prises par k.
Elle renvoie donc ainsi en sortie la factorielle du nombre n.

c. Proposer une fonction `factorielle2` qui réalise la même transformation que `factorielle`, mais en utilisant l'instruction `while` plutôt que `for`.

```
def factorielle2(n):
    f=1
    k=2
    while k<=n:
        f=f*k
        k+=1
    return f
```

2. Décrire le fonctionnement de la fonction `bin`.
La fonction `bin` prend en entrée deux paramètres entiers naturels `k` et `n`.
Elle calcule le coefficient binomial :

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

en faisant appel à la fonction `factorielle` créé dans le même listing.
Le quotient du calcul amène le langage Python à renvoyer un nombre de type `float`, alors qu'un coefficient binomial est un nombre entier. Le programme corrige le type en réaffectant la variable `c` à l'image de `c` par l'instruction/la fonction `int` avant de renvoyer le résultat en sortie.
Remarque : On aurait aussi pu économiser une ligne en écrivant : `return int(c)`.

3. Décrire le fonctionnement de la fonction `pasc`.
La fonction `pasc` prend en entrée un entier N, mais ne renvoie rien en sortie, elle réalise un affichage des N premières lignes du triangle de Pascal.
Elle active une boucle séquentielle où la variable n prend les valeurs entières de 0 à N.
A l'intérieur de cette boucle, une autre boucle séquentielle où la variable k prend les valeurs entières de 0 à n pour calculer (en faisant appel à la fonction `bin` créée plus haut dans le listing) et afficher sur une seule ligne la liste des coefficients binomiaux de la ligne n. A l'issue de cette boucle la fonction `pasc` réalise un retour à la ligne pour commencer au besoin la ligne suivante du triangle de Pascal.

Exercice 2

Créer une fonction qui réalise les tâches suivantes :

- prendre en entrée un mot de passe sous la forme d'une chaîne de caractères ;
- si ce mot de passe a un nombre pair de lettres, créer un nouveau mot de passe qui intervertit la première et la seconde moitié du mot de passe ;
- si ce mot de passe a un nombre impair de caractères, ajoute le caractère 'a' à la fin du mot de passe et réalise l'interversion ci-dessus.

Une solution possible :

```
def fonction(motdepasse):
    longueur = len(motdepasse)
    if (longueur%2)==0: # longueur est pair
        début=motdepasse[0:int(longueur/2)]
        fin=motdepasse[int(longueur/2):longueur]
        nouveau_motdepasse=fin+début
        return(nouveau_motdepasse)
    else: # longueur est impair puisqu'il n'est pas pair
        motdepasse=motdepasse+'a'
        longueur=longueur+1
        début=motdepasse[0:int(longueur/2)]
        fin=motdepasse[int(longueur/2):longueur]
        nouveau_motdepasse=fin+début
        return(nouveau_motdepasse)
```

Une solution qui évite d'écrire deux fois une partie du code :

```
def fonction(motdepasse):
    longueur = len(motdepasse)
    if (longueur%2)==1: # longueur est impair
        motdepasse=motdepasse+'a'
        longueur+=1
    début=motdepasse[0:int(longueur/2)]
    fin=motdepasse[int(longueur/2):longueur]
    nouveau_motdepasse=fin+début
    return(nouveau_motdepasse)
```

Exercice 3

Décrire les deux fonctions présentées dans le listing suivant :

```
def décomp(n):
    print('Décomposition en produit de facteurs premiers :')
    print(n,' = ',end='') # end='' c'est pour qu'il n'y ait pas de retour à ligne
    q=n # q sera les quotients entiers successifs
    for p in [2,3,5,7,11,13,17,19,23,29,31]:
        q=fact(p,q)
        if q==1:
            return()
    print(q,'.')
    if q>=37**2:
        print('ATTENTION : limite de ce programme, le dernier facteur n\'est peut-être pas premier. Il a un plus petit diviseur premier supérieur à 31.')

def fact(p,q):
# extrait autant de fois que possible le facteur p de q
    while q%p==0: # p divise q
        print(p,end='')
        q=int(q/p) # int pour éviter de passer en type float
        if q!=1:
            print(' x ',end='')
        else:
            print('.')
            return(q)
    return(q)
```

La fonction `décomp` prend en entrée un entier naturel n, mais ne retourne pas de donnée, elle produit un affichage à interpréter par l'utilisateur.
Elle réalise une boucle séquentielle où la variable p prend successivement les valeurs d'une liste qui sont les nombres premiers jusqu'à 31.
Dans cette boucle elle fait appel à la seconde fonction `fact`. Cette dernière prend en entrée deux variables : un nombre premier p et un entier naturel q.
Tant que p est un diviseur de q, elle divise q par p et affiche le facteur p ainsi que le signe multiplier.
Si $q = 1$, la décomposition en produit de facteurs premiers est achevée, et le programme affiche un point puis s'arrête en interrompant la boucle par l'instruction `return()`.

Quand p ne divise plus q et que q est différent de 1, la fonction `fact` renvoie à la fonction `décomp` le nombre q divisé autant de fois que possible par le nombre premier p.
La fonction `décomp` relance alors le processus avec le nombre premier suivant.
Si à l'issue du processus le nombre q est différent de 1 et inférieur au carré du plus grand nombre premier de la liste (31), cela implique qu'il est lui-même premier (s'il n'était pas premier, il aurait un diviseur premier inférieur ou égal à 31, ce qui est impossible d'après l'algorithme).
Dans le cas contraire, on est certain que q possède un plus petit diviseur premier supérieur au plus grand diviseur premier de la liste (il est peut-être lui même premier).

Remarque : On peut améliorer cette fonction en augmentant la liste des nombres premiers. Elle pourrait traiter plus de cas, mais resterait limitée.
Pour traiter tous les cas (sans recourir à une fonction mathématique particulière de bibliothèque publique) on peut créer une fonction qui génère automatiquement le nombre premier suivant un nombre donné.

$\sim$