Initiation à l'informatique - 1er semestre
TD 2

Exercice 2.1 *Puissance n-ème*
On veut calculer a^n pour un nombre réel a et un entier positif n.

1. Écrire deux versions de la fonction `puissance(a,n)`, la première version utilisera une boucle `tant que`, la deuxième une boucle `pour`.
2. Quelle est la complexité de vos fonctions ?
3. On veut maintenant calculer $\sum_{i=0}^{i=n} a^i$. Écrire une fonction `somme_puissances(a,n)` qui utilise la fonction `puissance`. Quelle est sa complexité ?
4. Écrire une deuxième version de la fonction `somme_puissances(a,n)`, qui n'utilise pas la fonction `puissance`.

Exercice 2.2
On définit la fonction `mystere (a,n)` suivante pour un nombre a quelconque non nul et un entier $n \geq 0$:

```
def mystere (a, n):
    if n == 0:
        return 1
    return a * mystere(a, n - 1)
```

1. Que renvoie cette fonction pour `a= 3` et `n= 4` ?
2. Que renvoie cette fonction dans le cas général ?

Exercice 2.3 *Factorielle*

1. Ecrire une fonction `factorielle(n)` qui retourne $n!$ à l'aide d'une boucle `pour`.
   Reécrire cette fonction à l'aide d'une boucle `tant que`.
   En déduire une fonction qui calcule $C_n^p = \frac{n!}{p! \times (n-p)!}$.
2. On définit la fonction `f (n)` suivante :

   ```
   def f (n):
       return n * f(n - 1)
   ```

   Que se passe-t-il si on appelle cette fonction avec $n = 3$?
   Corriger cette fonction pour qu'elle renvoie : $n!$

Exercice 2.4 *Nombres premiers*
L'objectif est d'écrire un algorithme qui retourne `vrai` si un entier naturel non nul n est premier et `faux` sinon.

1. Sachant qu'un nombre est premier s'il admet exactement deux diviseurs, une première approche consiste à dénombrer les diviseurs de n.
2. En observant, d'une part que l'on peut conclure dès qu'un diviseur autre que 1 ou n est trouvé, d'autre part qu'il est inutile de rechercher des diviseurs au-delà de $\sqrt{n}$, écrire un deuxième algorithme s'appuyant sur ces remarques.

Exercice 2.5

Écrire une fonction `pi_sur_4(n)` qui renvoie

$$\sum_{k=0}^{n} \frac{(-1)^k}{(2k+1)}$$

Exercice 2.6

Quelle la complexité de l'algortihme suivant :

```
def f(n):
  somme = 0
  for i in range (1,n+1):
    for j in range (i,n+1):
      somme = somme + i*j
  return somme
```

Exercice 2.7 *Jeu du nombre mystérieux*

1. On veut faire un jeu où le joueur doit deviner un entier déterminé aléatoirement par la machine et compris entre 0 et n. A chaque essai, l'algorithme précise si le nombre trouvé est "plus grand" ou "plus petit". À la fin du jeu le nombre d'essais est affiché. Imaginez une stratégie rustique de la part du joueur avant de l'améliorer.
2. Dans la version optimisée combien de propositions au maximum sont nécessaires pour découvrir un nombre si `n=1000` `n=100 000` ?