Devoir d'informatique n° 1 (25/11/2015) **Corrigé**

Cet énoncé est également le document réponse à rendre. Il comporte 5 exercices totalement indépendants. A la fin figure une feuille blanche qui vous servira de brouillon. Durée : 2h. Écrivez proprement votre code en indiquant bien les indentations.

Exercice 1 (*Questions indépendantes*)

Les commandes suivantes sont tapées dans la console. Indiquez ce que retourne chaque commande. Par exemple :

```
>>> 5*3
15
```

1. `len([1,4,5])`
2. `'alpha'+'bet'`
3. `type(2.0)`
4. `a=8 ; print(a*2)`

Exercice 1 :

```
>>>len([1,4,5])
3
>>>'alpha'+'bet'
'alphabet'
>>>type(2.0)
<type 'float'>
>>>a=8 ; print(a*2)
8
```

Exercice 2 (*Représentation graphique*)

1. Écrire une commande permettant de charger le module `numpy` sous le nom `np`.
2. On donne la fonction `mystere(x)` plus bas. Expliquer le fonctionnement de cette fonction.
3. Définir deux fonctions `g(x)` et `h(x)` qui retournent respectivement

$$g(x) = \cos^4(x) \quad \text{et} \quad h(x) = g(x) - \frac{1}{8}\cos(4x) - \frac{1}{2}\cos(2x) - \frac{3}{8}.$$

4. Écrire une fonction Python `graph(a,b,p)` qui représente sur l'intervalle $[a, b]$ et avec p points la fonction h sur un graphique.
5. Déterminer une primitive de $\cos^4$.
6. On exécute dans la console `>>> graph(2,4,1000)`. Qu'obtient-t-on ?

Exercice 2 :

Question 1 :

```
import numpy as np
```

Question 2 :

```
def mystere(x):
    if x == 0:
        raise ZeroDivisionError
    else:
        return 1.0/x
```

Explications : La fonction retourne le flottant $1/x$ lorsque $x \neq 0$ et soulève une erreur de division par 0 lorsque x est nul.

Question 3 :

```
def g(x):
   return np.cos(x)**4

def h(x):
   return g(x) - 1.0/8*np.cos(4*x)-1.0/2*np.cos(2*x)-3.0/8
```

Question 4 :

```
def graph(a,b,p):
   abscisse = np.linspace(a,b,p)
   ordonnee = [h(x) for x in abscisse]
   return plot(abscisse,ordonnee)
```

Question 5 : Détermination d'une primitive de $\cos^4$.
Pour tout $x \in \mathbb{R}$, on a :

$$cos^4 x = \left(\frac{e^{ix}+e^{-ix}}{2}\right)^4 = \frac{1}{16}\left[e^{4ix}+e^{-4ix}+4e^{2ix}+4e^{-ix}+6\right] = \frac{1}{8}\cos(4x)+\frac{1}{4}\cos(2x)+\frac{3}{8}.$$

Ainsi, $\cos^4$ étant continue sur $\mathbb{R}$, elle admet une primitive sur $\mathbb{R}$ donnée par

$$x \mapsto \frac{1}{32}\sin(4x)+\frac{1}{8}\sin(2x)+\frac{3x}{8}.$$

Question 6 : On obtient le graphe de la fonction nulle sur $[2,4]$ (en réalité avec les erreurs d'arrondis, le graphe n'est pas exactement une droite).

Exercice 3 (*Des comparaisons de suites*)

On rappelle que pour obtenir la valeur absolue, la commande Python est `abs(a)` où a est la nombre dont on veut obtenir la valeur absolue et que `exp(1)` permet d'avoir al valeur de e.

1. Écrire une fonction `facto(n)` qui renvoie $2 \times 4 \times \cdots \times (2n)$ pour un entier n.
2. Réécrire la fonction `facto(n)` précédente et qui renverra une erreur `TypeError` lorsque n n'est pas un entier avec le message « `l argument n est pas un entier` » .
3. Écrire une fonction `suite(n)` qui renvoie $v_n = \sqrt{2\pi n}\left(\frac{2n}{e}\right)^n$.
4. Écrire une fonction `repres(n)` qui représente graphiquement les termes $u_1, \cdots, u_n$ où pour tout entier $k \in \mathbb{N}^*$:
$$u_k = \frac{2 \times 4 \times \cdots \times (2k)}{v_k}.$$
5. Écrire une fonction `plus_petit_element(P)` qui retourne le plus petit entier n tel que
$$|u_k - 1| \leq P.$$

Exercice 3 :

Question 1 :

```
def facto(n):
   S=1
   for i in range(1,n+1):
      S = S*2*i
   return S
```

Question 2 :

```
def facto(n):
   if type(n) == int:
      S=1
      for i in range(1,n+1):
         S = S*2*i
      return S
   else:
      raise TypeError('l argument n est pas un entier')
```

Question 3 :

```
def suite(n):
   return sqrt(2*pi*n)*((2*n)/exp(1))**n
```

Question 4 :

```
def repres(n):
   abscisse = np.linspace(1,n,n)
   ordonne = [float(facto(k))/suite(k) for k in range(1,n+1)]
   return plot(abscisse,ordonne,'o')
```

Question 5 :

```
def plus_petit_element(P):
   k=1
   while abs(float(facto(k))/suite(k)-1)> P:
      k=k+1
   else:
   return k
```

Exercice 4 (*Des suites récurrentes*)

1. Écrire une fonction `test(a,b)` qui retourne `1` si $a \leq b$ et `0` sinon.
2. Écrire une fonction `suites(n,a,b)` qui renvoie une liste avec deux éléments où le premier élément est a_n et le second élément est b_n où on pose $a_0 = a, b_0 = b \in \mathbb{R}^+$ avec $b \geq a$ et pour tout $n \in \mathbb{N}$:

$$a_{n+1} = \sqrt{a_n b_n} \quad \text{et} \quad b_{n+1} = \frac{a_n + b_n}{2}.$$

3. Réécrire la fonction précédente pour que celle-ci retourne le message `'a <= b est faux'` lorsque $a \leq b$ n'est pas vérifié.
4. On admet que pour tout $n \in \mathbb{N}$, $a_n \leq b_n$. Montrer que (a_n) est croissante et (b_n) décroissante.

Exercice 4 :

Question 1 :

```
def test(a,b):
   if a <= b:
      return 1
   else:
      return 0
```

Question 2 :

```
def suites(n,a,b):
   L = [a,b]
   if n==0:
      return L
   else:
      for i in range(1,n+1):
         L[0],L[1] = sqrt(L[0]*L[1]), float(sum(L))/2
      return L
```

Question 3 :

```
def suites(n,a,b):
   if test(a,b) == 1:
      L = [a,b]
      if n==0:
         return L
      else:
         for i in range(1,n+1):
            L[0],L[1] = sqrt(L[0]*L[1]), float(sum(L))/2
         return L
   else:
      print('a <= b est faux')
```

Question 4 : Montrer que (a_n) est croissante et (b_n) décroissante.
On a, pour tout $n \in \mathbb{N}$:

$$\frac{a_{n+1}}{a_n} = \sqrt{\frac{b_n}{a_n}} \geq 1$$

et

$$b_{n+1} - b_n = \frac{a_n - b_n}{2} \leq 0$$

car $\sqrt{}$ est croissante et $a_n \leq b_n$. Ainsi, (a_n) est décroissante et (b_n) est croissante.

Exercice 5 (*Diviseurs*)

On rappelle que la commande Python `a%b` renvoie la reste de la division euclidienne de a par b et que b est un diviseur de a lorsque ce reste est nul.

1. Écrire une fonction `diviseurs(n)` qui renvoie la liste des diviseurs de n.
2. On donne la fonction `mystere(a,b)` plus bas. Expliquer le rôle de cette fonction puis réécrire cette fonction en créant une documentation.
3. À l'aide de la question 1 ou de la question 2, écrire une fonction `PGCD(a,b)` qui retourne le plus grand diviseur commun de a et b.
4. Convertir le nombre $\overline{100010}^2$ en écriture et $\overline{245}^{10}$ en binaire. On expliquera dans chaque cas la méthode employée.
5. Écrire une fonction `deci_bin(a)` qui a partir d'un nombre a écrit en décimal retourne une liste correspondant à l'écriture en binaire de a. On pourra utiliser la commande `L.insert(0,i)` qui insère en première position l'élément i dans la liste `L`.

Exercice 5 :

Question 1 :

```
def diviseurs(n):
   L=[]
   for k in range(1,n+1):
      if n%k == 0:
         L.append(k)
      else:
         pass
   return L
```

Question 2 :

```
def mystere(a,b):
   L=[]
   for i in range(1,min(a,b)+1):
      if a%i == 0 and b%i == 0:
         L.append(i)
      else:
         pass
   return L
```

Explications : La fonction retourne la liste des diviseurs communs à a et b.
Réécriture de la fonction :

```
def mystere(a,b):
   '' '' '' Retourne la liste des diviseurs commun a a et b

   Exemple :
   >>>diviseurs(25,15)
   [1,5] '' '' ''
   L=[]
   for i in range(1,min(a,b)+1):
      if a%i == 0 and b%i == 0:
         L.append(i)
      else:
         pass
   return L
```

Question 3 :

```
def PGCD(a,b):
   return max(mystere(a,b))
```

Question 4 : Convertir le nombre $\overline{100010}^2$ en écriture et $\overline{245}^{10}$ en binaire. On a :

$$\overline{100010}^2 = 1 \times 2^1 + 1 \times 2^5 = 2 + 32 = 34$$

et en effectuant des divisions euclidiennes successives :

$$\overline{245}^{10} = 1 + 2^2\Big(2^2\big(1 + 2(1 + 2[1 + 2])\big)\Big) = 2^0 + 2^2 + 2^4 + 2^5 + 2^6 + 2^7 = \overline{11110101}^2.$$

Question 5 :

```
def deci_bin(a):
   L=[]
   if a == 0:
      return [0]
   else:
      while a > 0:
         L.insert(0,a%2)
         b=a//2
         a=b
   return L
```