TP 2 : Boucles `for` et `while`

I Boucle `for`

Exercice 1. Écrire un script qui calcule le n-ième terme d'une suite géométrique de raison 2 et de premier terme 3, ainsi que la somme des $n+1$ premiers termes de cette suite.

Réponse :

```
def u(n):
    #En supposant que u_0=3 et u_{n-1} est le n_ième terme
    u=3
    for i in range (1,n-1):
        u=2*u
    return u

def S(n):
    u=3
    S=u
    for i in range (1,n):
        u=2*u
        S=S+u
    return S
```

Exercice 2. Écrire une fonction `produit` qui prend en argument deux entiers naturels $n > 0$ et p et qui calcule (et renvoie) le produit $P = \prod_{k=1}^{p} \frac{n+1-k}{n}$ si $p \in [\![1, n]\!]$ et qui renvoie `"Il faut que p soit compris entre 1 et n"` sinon.

Réponse :

```
def produit(n,p):
    if p>n or p<1 :
        return "Il faut que p soit compris entre 1 et n"
    else :
        P=1
        for k in range(1,p+1):
            P=P*((n+1-k)/n)
        return P
```

Exercice 3. Pour $n \in \mathbb{N}^\star$, on pose

$$R(n) = \sqrt{1 + \sqrt{2 + \cdots + \sqrt{n-1+\sqrt{n}}}}.$$

Écrire une fonction qui prend en argument un entier n et qui renvoie $R(n)$.

Réponse :

```python
def R(n):
    R=0
    for i in range(n,0,-1):
        R=sqrt(i+R)
    return R

def R(n):
    R=0
    for i in range(n):
        R=sqrt(n-i+R)
    return R
```

Exercice 4. Écrire un script Python qui prend en argument un nombre naturel $n \in \mathbb{N}^*$ et permet de calculer le terme général des suites suivantes :

$$a_n = \left(\sum_{k=1}^{n} \frac{1}{k}\right)^2, \qquad b_n = \sum_{k=1}^{n} \frac{1}{k^2} \qquad \text{et} \qquad c_n = \sum_{k=1}^{n^2} \frac{1}{k}$$

Réponse :

```python
def a(n):
    S=0
    for k in range(1,n+1):
        S=S+(1/k)
    S=S**2
    return S
```

```python
def b(n):
    S=0
    for k in range(1,n+1):
        S=S+(1/k**2)
    return S
```

```python
def c(n):
    S=0
    for k in range(1,n**2+1):
        S=S+(1/k)
    return S
```

Exercice 5. La suite de Fibonacci est la suite définie par $u_0 = u_1 = 1$ et pour tout $n \in \mathbb{N}$, $u_{n+2} = u_{n+1} + u_n$.

Écrire une fonction `Fibonacci` qui prend en argument un entier n et qui renvoie le n-ième terme de la suite de Fibonacci.

Réponse :

```python
#Avec 3 variables
def Fibonacci(n):
    u0=u1=1
    u=1
    for i in range(n-2):
        #on stocke la valeur de u1 dans la variable u avant sa modification
        u=u1
        u1=u1+u0
        u0=u
    return u1
```

```python
#Avec l'affectation parallèle
def Fibonacci(n):
    u0=u1=1
    for i in range(n-2):
        u1,u0=u1+u0,u1
    return u1
```

II Boucles imbriquées

Exercice 6. 1. Écrire une fonction `somme1` qui prend en argument deux entiers j et n et renvoie la somme $\sum_{k=1}^{n} k^j$.

2. Utiliser la fonction précédente pour calculer la somme $S = \sum_{j=1}^{n} \sum_{k=1}^{n} k^j$ pour $n = 10$.

3. Donner une formule simple pour la somme $\sum_{j=1}^{n} k^j$. On distinguera bien le cas $k = 1$ et $k \neq 1$.

4. En déduire une fonction `somme2` qui prend en argument deux entiers k et n et qui renvoie la somme $\sum_{j=1}^{n} k^j$ sans utiliser de boucle.

5. Utiliser la fonction précédente pour recalculer la somme S.

Réponse :

```
#Question 1)
def somme1(j,n):
    S=0
    for k in range(1,n+1):
        S=S+k**j
    return S
```

```
#Question 2)
S=0
for j in range(1,11):
    S=S+somme1(j,10)
print(S)
```

Question 3)

Si $k = 1$ alors $\sum_{j=1}^{n} k^j = n$. Si $k \neq 1$ alors $\sum_{j=1}^{n} k^j = k \times \frac{1 - k^n}{1 - k}$

```
#Question 4)
def somme2(k,n):
    if k==1:
        S=n
    elif k>1 :
        S=k*(1-k**n)/(1-k)
    return S
```

```
#Question 5)
S=0
for k in range(1,11):
    S=S+somme2(k,10)
print(S)
```

Exercice 7. Écrire un script Python qui prend en argument un nombre naturel $n \in \mathbb{N}^*$ et permet de calculer le terme général des suites suivantes :

$$a_n = \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{i - j}{i + j}, \qquad b_n = \sum_{k=1}^{n} \sum_{j=1}^{k} \frac{\cos(k)}{j^2} \qquad \text{et} \qquad c_n = \sum_{i=1}^{n} \sum_{j=i}^{n} \min(i, j)$$

Réponse :

```python
def a(n):
    S=0
    for i in range(1,n+1):
        for j in range(1,n+1):
            S=S+((i-j)/(i+j))
    return S

def b(n):
    S=0
    for k in range(1,n+1):
        for j in range(1,k+1):
            S=S+(cos(k)/j**2)
    return S

def c(n):
    S=0
    for i in range(1,n+1):
        for j in range(i,n+1):
            S=min(i,j)
    return S
```

III Boucle `while`

Exercice 8. Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_0 = \dfrac{1}{2}$ et $u_{n+1} = u_n^2 - u_n + 1$ pour tout $n \in \mathbb{N}$. On peut montrer que cette suite tend vers $\ell = 1$.

On souhaite écrire un script qui calcule les termes de la suite $(u_n)_{n\in\mathbb{N}}$ jusqu'à ce que u_n soit proche de sa limite à 10^{-4} près, puis qui affiche le dernier terme de la suite calculé ainsi que le nombre de termes qu'il a fallu calculer.

1. Quel sont le test d'arrêt et test d'exécution.
2. Écrire le script.

Réponse :

1. Le test d'arrêt est $|u_n - l| \leq 10^{-4}$ Le test d'exécution est $|u_n - l| > 10^{-4}$
2.

```
u=1/2
n=0
l=1
while u-l>=0.0001 or l-u>=0.0001:
    u=u**2-u+1
    n=n+1
print(u,n)
```

Exercice 9. Étude de la série harmonique alternée.

On considère la suite $(u_n)_{n\geq 1}$ définie par : $\forall n \in \mathbb{N}^\star,\ u_n = \displaystyle\sum_{k=1}^{n} \frac{(-1)^{k+1}}{k}$.

On peut montrer que cette suite converge vers une limite S, et que $\forall n \in \mathbb{N}^* : |u_n - S| \leq |u_n - u_{n-1}| \leq \dfrac{1}{n}$.

1. Écrire une fonction qui calcule S à 10^{-4} près.
2. On peut montrer que $S = \ln(2)$. Vérifier le résultat obtenu en comparant la valeur trouvée cette valeur.

Réponse :

On a $|u_{10^4} - S| \leq \frac{1}{10^4}$. Donc u_{10^4} se trouve dans un voisinage de S de rayon 10^{-4}. Donc en calculant, u_{10^4}, on calcule S à 10^{-4} près :

```
u=0
for k in range(1,10001):
    u=u+((-1)**(k+1)/k)
print(u)

log(2)-u
```

La commande `log(2)-u` affiche `4.999749998702008e-05`

Exercice 10. Un peu de pliage
On plie plusieurs fois une feuille de papier de format A4 (21 cm × 29.7 cm) et d'épaisseur 0.01 cm, et on veut calculer les dimensions de cette feuille après un certain nombre de pliages. Les pliages sont faits de façon à plier en deux la feuille toujours selon la plus grande dimension.

1. Écrire un script qui prend en argument le nombre n de pliages qu'il souhaite effectuer, puis qui renvoie les dimensions (longueur, largeur et épaisseur) de la feuille après n pliages. Tester pour 5 pliages, puis pour 10 pliages.
2. Écrire un nouveau script qui prend en argument la hauteur h en cm, qui calcule combien de pliages sont nécessaires pour que l'épaisseur finale du papier soit supérieure à h, et qui affiche les dimensions de la feuille après ces pliages. Tester pour une hauteur de 2.5 m, puis pour la distance Terre-Lune (environ 380 400 km).

Réponse :

```
def pliages(n):
    L=21
    l=29.7
    e=0.01
    for i in range(n):
        e=2*e
        if L<l:
            l=l/2
        else :
            L=L/2
    return (L,l,e)

def hauteur(h):
    L=21
    l=29.7
    e=0.01
    n=0
    while e<h:
        e=2*e
        n=n+1
        if L<l:
            l=l/2
        else :
            L=L/2
    return n
```

Les commandes `hauteur(250)` et `hauteur(38040000000)` affichent respectivement : `15 et 42`

Exercice 11. Conjecture de Syracuse
L'algorithme de Syracuse consiste à itérer l'opération suivante : à un nombre entier n, on associe $\dfrac{n}{2}$ si n est pair et $3n+1$ si n est impair. On conjecture (on ne sait toujours pas si c'est vrai) que quel que soit l'entier considéré initialement dans cet algorithme, on arrive toujours à 1 après un certain nombre d'itérations. C'est en tout cas vrai pour tous les entiers avec lesquels l'algorithme a été testé.
Écrire un programme qui prend un entier n, effectue l'algorithme de Syracuse, puis affiche tous les nombres obtenus jusqu'au premier 1 et donne le nombre d'itérations effectuées jusqu'à l'obtention du premier 1. Le tester sur différentes valeurs.

Réponse :

```
def Syracuse(n):
    while n!=1:
        if n%2==0:
            n=n/2
            print(n)
        else :
            n=3*n+1
            print(n)
```