

Exercice 1

Suite récurrente et fonction recursive**1.1 Script récursif**

Donner dans chacun des cas suivants, le script récursif qui calcule le terme de rang n de la suite :

a. $u_n = 2 \times u_{n-1} + 1, u_0 = 0$

b. $v_n = \frac{1}{v_{n-1}}, v_0 = 1$

c. $f_n = n \times f_{n-1}$

d. $fb_n = fb_{n-1} + fb_{n-2}$

1.2 Script itératif

Pour la série (u_n) de la question a., donner le script de la fonction itérative qui calcule le terme de rang n de la suite

Exercice 2

Fonctions pair / impair

On donne le script de la fonction pair :

```
1 def pair(N):
2     if N==0 :
3         return True
4     else :
5         return pair(N-2)
```

- 2.1 Que renvoie pair(4) ? Expliquez en faisant le **tracé** du résultat (représenter la pile d'appels).
- 2.2 Comment se comporte le programme si on fait : pair(5) ?
- 2.3 Modifier alors le script proposé pour tenir compte de ce problème (il faudra modifier la condition de base).
- 2.4 Ecrire le script de la fonction impair, qui renvoie True si l'argument est ...impair.

Exercice 3

Fonction mystère

```
1 def f(x,y):
2     if x == y:
3         return x
4     elif x < y:
5         return f(x,y-x)
6     else:
7         return f(x-y,y)
```

3.1 Sans exécuter le code, indiquer ce que renvoient $f(5, 15)$ et $f(8, 29)$

3.2 Indiquer plus généralement ce que calcule $f(x, y)$.

Exercice 4

Les vaches de Narayana

Une vache de plus de 3 ans, donne naissance à une autre tous les ans, en début d'année, qui elle-même donne naissance à une autre chaque année à partir de sa quatrième année (elle a 3 ans et 1 jour).

Partant d'une vache qui vient de naître ($v_1 = 1$), les termes successifs de la suite (v_i) , donnant le nombre de vaches, sont donc :

$$1, 1, 1, 2, 3, 4, 6, \dots$$

4.1 Écrire une fonction récursive $v(i)$ qui renvoie le i -ème terme de la suite.

4.2 La complexité est-elle exponentielle ou linéaire ?

Exercice 5

La fonction de Mc Carthy

La fonction 91 de McCarthy est une fonction récursive définie par McCarthy dans son étude de propriétés de programmes récursifs, et notamment de leur vérification formelle. (https://fr.wikipedia.org/wiki/Fonction_91_de_McCarthy)

C'est une fonction dont l'image est égale à 91 pour tout entier $n < 102$.

Elle est définie pour tout $n \in \mathbb{N}$.

$$f(n) = \begin{cases} n - 10 & \text{si } n > 100 \\ f(f(n + 11)) & \text{sinon} \end{cases}$$

5.0.1 Écrire en python le script de cette fonction recursive

5.0.2 Représenter la pile d'appels pour $f(99)$ par cette fonction. Converge t-elle bien vers 91 ?

Exercice 6

Prolongement : Une fonction $x + 1/x$

Considérons la suite u_n

$$n \in \mathbb{N}$$

définie par

$$u_0 = 1$$

et la relation de recurrence :

$$u_{n+1} = u_n + \frac{1}{u_n}$$

6.1 Compléter le script récursif de cette fonction que l'on nommera `u_rec`. Ecrire également son docstring (entre guillemets, au début).

6.2 Complexité

On utilisera les conventions suivantes pour le calcul de la complexité :

Opérations	Poids
$+$, $-$, \times , \div	1 unité de temps
Affectation	1 unité de temps
Appel de fonction	1 unité de temps
Comparaison	1 unité de temps

6.2.1 Quelle est la loi de recurence sur le nombre d'instructions $T(n)$ en fonction de $T(n-1)$ pour cette fonction.

6.2.2 En déduire la complexité asymptotique $O(g(n))$.

6.3 Dans votre script, où pourrait-on ajouter un test d'assertion pour protéger la fonction d'une entrée non conforme (par exemple avec $n < 0$) ? Ecrire l'instruction de ce test d'assertion.

6.4 On propose un autre script pour cette fonction :

```

1 def u_rec (n) :
2     if n==0:
3         return 1
4     else :
5         x=u_rec (n-1) # variable locale
6         return x+1/x

```

6.4.1 Cette fonction, est-elle plus efficace ? C'est à dire, est-elle de complexité inférieure ? Justifiez rapidement.

Exponentiation

Etudions l'exponentiation à travers deux exemples.

```
1 def exp1(n,x) :  
2     """  
3     programme qui donne x^n en sortie sans utiliser **  
4     n : entier  
5     x : reel  
6     exp1 : reel  
7     """  
8     acc=1  
9     for i in range(1,n+1):  
10         acc*=x  
11     return acc  
12  
13 def exp2(n,x):  
14     """  
15     n : entier  
16     x : reel  
17     exp2 : reel  
18     """  
19     if n==0 : return 1  
20     else : return exp2(n-1,x)*x
```

1. Combien de produits sont nécessaires pour calculer une puissance n -ième avec la fonction `exp1` ?
2. Pour la fonction `exp2` : Soit u_n le nombre de produits nécessaires pour calculer une puissance n -ième. Quelle est la relation de récurrence vérifiée par u_{n+1} ?

$$u_{n+1} = \dots$$

3. En déduire la complexité pour ces 2 fonctions.

Partie 2

Exercice 8

Les tours de Hanoï

Voir le cours en ligne sur la complexité. L'exemple y est longuement traité.

8.1 Principe

On considère trois tiges plantées dans une base. Au départ, sur la première tige sont enfilées N disques de plus en plus étroits. Le but du jeu est de transférer les N disques sur la troisième tige en conservant la configuration initiale.

8.2 algorithme récursif

L'algorithme récursif pour ce problème est étonnamment réduit :

```
1 def hanoi(N,d,i,a):
2     """N disques doivent être déplacés de d vers a
3     Params:
4     N : int
5         nombre de disques
6     d: int
7         depart (vaut 1 au debut)
8     i: int
9         intermediaire (vaut 2 au debut)
10    a: int
11        fin (vaut 3 au debut)
12    Exemple:
13    lancer avec
14    >>> hanoi(3,1,2,3)
15    """
16    if N==1 :
17        print('deplacement de {} vers {}'.format(d,a))
18    else:
19        hanoi(N-1,d,a,i)
20        hanoi(1,d,i,a)
21        hanoi(N-1,i,d,a)
```

Résultat

```
1 >>> hanoi(3,1,2,3)
2 deplacement de 1 vers 3
3 deplacement de 1 vers 2
4 deplacement de 3 vers 2
5 deplacement de 1 vers 3
6 deplacement de 2 vers 1
7 deplacement de 2 vers 3
8 deplacement de 1 vers 3
```

- 8.2.1 Vérifier que pour $N = 2$ disques, il y a 3 déplacements, que pour 3 disques, il y en a 7, et que pour 4 disques, il y en a 15.
- 8.2.2 Proposez une loi de récurrence entre le nombre de déplacements $T(N)$ pour N disques, et le nombre de déplacements $T(N-1)$ pour $N-1$ disques.
- 8.2.3 Cette loi, est-elle conforme à celle que l'on aurait déduite de l'étude de la complexité pour l'algorithme récursif?

Exercice 9

Dessin récursif

Soit la relation de récurrence suivante :

$$u_{n+1} = \frac{8u_n}{9} + \frac{1}{9}$$

Le premier terme de la suite est $u_0 = 0$

9.1 Script python

Ecrire en python le script de la fonction recursive correspondante, qui calcule au rang n le nombre u_n . Appeler cette fonction f .

9.2 Application géométrique

On considère un carré de côté 1. On le partage en 9 carrés égaux, et on colorie le carré central. Puis, pour chaque carré non-colorié, on réitère le procédé. On note u_n l'aire coloriée après l'étape n . La suite u_n est $u_{n+1} = \frac{8u_n}{9} + \frac{1}{9}$

Quelle est la limite de la suite u_n ?

Exercice 10

Longueur d'une liste

10.1 algorithme itératif :

La fonction suivante calcule la longueur d'une chaîne de caractères `seq` passée en argument.

```
1 def len_iterative(seq):
2     """
3     Return the length of a list (iterative)
4     """
5     count = 0
6     for elt in seq:
7         count = count + 1
8     return count
```

Réaliser la preuve de cet algorithme.

10.2 algorithme récursif

- Montrer que la relation de récurrence $u_{n+1} = 1 + u_n$ permet de compter de 1 en 1.
- Soit la chaîne de caractères `s = "abcd"`. On veut supprimer le premier caractère de `s` à l'aide d'une *slice* python. Quelle est l'instruction python correspondante ?

c. Ecrire le script python de l'algorithme récursif

Aide pour l'écriture de l'algorithme récursif : la fonction récursive s'appellera `len_recursive`, et aura pour argument `seq`. Si on veut passer en argument la liste `seq` de laquelle on retire le premier élément, on fait : `len_recursive(seq[1:])`. Il faudra alors s'inspirer de la relation de récurrence suivante lors de l'appel récursif :

$$u_{n+1} = 1 + u_n$$

Exercice 11

Algorithme de calcul du PGCD d'Euclide

11.1 Historique

En mathématiques, l'algorithme d'Euclide est un algorithme qui calcule le plus grand commun diviseur (PGCD) de deux entiers, c'est-à-dire le plus grand entier qui divise les deux entiers, en laissant un reste nul.

l'algorithme d'Euclide est l'un des plus anciens algorithmes. Il est décrit dans le livre VII (Proposition 1-3) des *Éléments* d'Euclide (vers 300 avant JC). Cela correspond à une adaptation de la méthode naïve de calcul de la division euclidienne. (*source wikipedia*)

11.2 Principe

- Il s'agit de divisions en cascade de A par B : les résultats de l'une servent à poser la suivante :
 - Le diviseur B devient le dividende ; et
 - Le reste r_1 devient le diviseur.
- Arrêt lorsque le reste de la division est nul.
- Le reste trouvé avant le reste nul est le PGCD des nombres A et B.

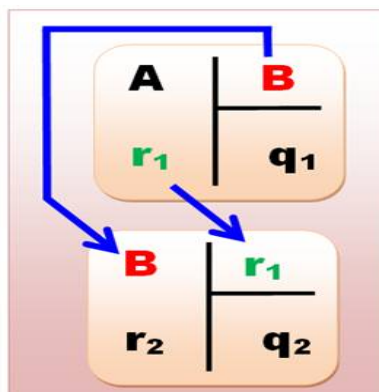


FIGURE 1 – division euclidienne

11.2.1 Compléter le tableau suivant montrant le tracé du calcul du PGCD de 264 par 228 :

division de A	par B	q	r
...			
...			
...			

11.2.2 Ecrire le script en python correspondant à la programmation itérative de cet algorithme.

```
1 def PGCD_it(a,b):
2     """
3     a et b sont des entiers, a > b
4     euclide retourne un entier qui est le PGCD de a et b
5     """
6
7     ...
8     ...
9     ...
10    ...
11    ...
12    ...
```

11.3 script récursif

```
1 def PGCD(a,b):
2     """
3     PGCD : entier correspondant au plus grand diviseur commun de a par b
4     a et b : entiers tels que a > b
5     """
6     if b == 0 : return a
7     else:
8         c = a % b
9         return PGCD(b,c)
```

11.3.1 Tracer le calcul de PGCD(264,228) en représentant la pile des appels successifs.**11.3.2 Prouver la terminaison de cette fonction recursive.**