Exercice 1

Suite récurente et fonction recursive

1.1 Script recursif

1. Donner dans chacun des cas suivants, le scrip 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. $f_n = n \times f_{n-1}$, $f_0 = 1$

d. $fb_n = fb_{n-1} + fb_{n-2}$, $fb_0 = 0$, $fb_1 = 1$

e. $w_n = w_{n-1} + \frac{1}{w_{n-1}}$

2. Calculer les 4 premiers termes de la suite

3. Quelles fonctions sont de classe de complexité linéaire ? Quadratique ? Exponentielle ?

1.2 Script itératif

Pour chacune des suites proposées, donner le script de la fonction **itérative** qui calcule le terme de rang n de la suite

Exercice 2

Fonction mystère

```python
def f(x,y):
  if x == y:
    return x
  elif x < y:
    return f(x,y-x)
  else:
    return f(x-y,y)
```

2.1 Faire le tracé de la fonction en utilisant un tableau de suivi des variables et indiquer ce que renvoient f(5, 15)

appel avec x =	y =	return
5	15	f(5,10)
5	10	...
..	..	..
..	..	..

2.2 Même question pour f(8, 29)

appel avec x =	y =	return
8	29	f(8,21)

appel avec x =	y =	return
8	21	...
..	..	..
..	..	..
..	..	..
..	..	..
..	..	..

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

Exercice 3

Les vaches de Narayana

Une vache de plus de 3 ans, donne naissance à une autre tous les ans, en debut 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 naitre ($v_1 = 1$), les termes successifs de la suite (v_i), donnant le nombre de vaches, sont donc :

$$1, 1, 1, 2, 3, 4, 6, ...$$

3.1 Donner les 3 termes suivants de cette suite.

3.2 Ecrire une fonction récursive v(i) qui renvoie le i-eme terme de la suite.

3.3 La complexité est-elle exponentielle ou linéaire ?

Exercice 4

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 $n \in \mathbb{N}$.

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

4.0.1 Ecrire en python le script de cette fonction recursive

4.0.2 Faire le tracé de la fonction en représentant la pile d'appels pour f(99) par cette fonction. Converge t-elle bien vers 91 ?

```
f(99) = f(f(110))
        = f(100)
        = f(...
```

Exercice 5

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 :

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

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

5.2 Compléxité

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

Opérations	Poids
+, −, ×, ÷	1 unité de temps
Affectation	1 unité de temps
Appel de fonction	1 unité de temps
Comparaison	1 unité de temps

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

5.2.2 En déduire la complexité en notation de Landau. Définir la classe de complexité.

5.3 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.

5.4 On propose un autre script pour cette fonction :

```
def u_rec (n) :
    if n==0:
        return 1
    else :
        x=u_rec (n-1) # variable locale
        return x+1/x
```

5.4.1 Cette fonction, est-elle plus efficace ? C'est à dire, est-elle de complexité infèrieure ? Justifiez.

Exercice 6

Longueur d'une liste

6.1 algorithme itératif :

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

```python
def len_iterative(seq):
    """
    Return the length of a list (iterative)
    """
    count = 0
    for elt in seq:
        count = count + 1
    return count
```

Réaliser la **preuve** de cet algorithme. C'est à dire montrer que ce programme va bien retourner la longueur de la chaine de caractères.

6.2 algorithme récursif

a. Montrer que la relation de récurence $u_{n+1} = 1 + u_n$ permet de compter de 1 en 1.

b. Soit la chaine de caractères `s = "abcd"`. On veut supprimer le premier caractère de `s` à l'aide d'un *slice* python. Quelle est l'instruction python correspondante ?

c. Ecrire le script python de l'algorithme récursif

Aide pour l'écriture de l'algorithme recursif : la fonction récursive s'appelera `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 récurence suivante lors de l'appel recursif :

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

d. Prouver la terminaison de l'algo recursif.

Exercice 7

Les tours de Hanoï

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

7.1 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.

7.2 algorithme récursif

L'algorithme récursif pour ce problème est étonnament réduit :

```python
def hanoi(N,d,i,a):
    """N disques doivent être déplacés de d vers a
    Params:
    N : int
        nombre de disques
    d: int
        depart (vaut 1 au debut)
    i: int
        intermediaire (vaut 2 au debut)
    a: int
        fin (vaut 3 au debut)
    Exemple:
    lancer avec
    >>> hanoi(3,1,2,3)
    """
    if N==1 :
        print('deplacement de {} vers {}'.format(d,a))
    else:
        hanoi(N-1,d,a,i)
        hanoi(1,d,i,a)
        hanoi(N-1,i,d,a)
```

Résultat

```
>>> hanoi(3,1,2,3)
deplacement de 1 vers 3
deplacement de 1 vers 2
deplacement de 3 vers 2
deplacement de 1 vers 3
deplacement de 2 vers 1
deplacement de 2 vers 3
deplacement de 1 vers 3
```

7.2.1 Vérifier (experimentalement) 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.

7.2.2 Proposez une loi de recurence entre le nombre de déplacements T(N) pour N disques, et le nombre de déplacements T(N-1) pour N-1 disques.

7.2.3 Retrouver la loi T(n) en fonction de T(n-1) en analysant le script de la fonction.

Exercice 8

Sujet Métropole Sept 2 2021 Exercice 4

Cet exercice porte sur la programmation en général et la récursivité en particulier.

On s'intéresse dans cet exercice à un algorithme de mélange des éléments d'une liste.

1. Pour la suite, il sera utile de disposer d'une fonction echange qui permet d'échanger dans une liste lst les éléments d'indice `i1` et `i2`.

Expliquer pourquoi le code Python ci-dessous ne réalise pas cet échange et en proposer une modification.

```python
def echange(lst, i1, i2):
 lst[i2] = lst[i1]
 lst[i1] = lst[i2]
```

2. La documentation du module random de Python fournit les informations ci-dessous concernant la fonction `randint(a,b)` :

```
Renvoie un entier aléatoire N tel que a <= N <= b.
```

Parmi les valeurs ci-dessous, quelles sont celles qui peuvent être renvoyées par l'appel `randint(0, 10)` ?

```
0   1   3.5   9   10   11
```

3. Le mélange de Fischer Yates est un algorithme permettant de permuter aléatoirement les éléments d'une liste. On donne ci-dessous une mise en œuvre récursive de cet algorithme en Python.

```python
from random import randint

def melange(lst, ind):
  print(lst)
  if ind > 0:
    j = randint(0, ind)
    echange(lst, ind, j)
    melange(lst, ind-1)
```

a. Expliquer pourquoi la fonction melange se termine toujours.

b. Lors de l'appel de la fonction melange, la valeur du paramètre `ind` doit être égale au plus grand indice possible de la liste `lst`.

Pour une liste de longueur `n`, **quel est le nombre d'appels récursifs de la fonction melange effectués, sans compter l'appel initial ?**

c. On considère le script ci-dessous :

```python
lst = [v for v in range(5)]
melange(lst, 4)
```

On suppose que les valeurs successivement renvoyées par la fonction `randint` sont 2, 1, 2 et 0.

Les deux premiers affichages produits par l'instruction `print(lst)` de la fonction melange sont :

```
[0, 1, 2, 3, 4]
[0, 1, 4, 3, 2]
```

Donner les affichages suivants produits par la fonction melange.

d. Proposer une version itérative du mélange de Fischer Yates.

On rappelle que la fonction `range` accepte 1 à 3 paramètres :

- avec un seul paramètre n, le variant prend successivement les valeurs de tous les entiers compris entre 0 et n-1
- avec 3 paramètres, le variant i prend toutes les valeurs entières comprises entre le 1er (inclus) et 2e paramètre (exclus). Le dernier paramètre sert à préciser le sens (croissant ou décroissant). Par exemple :

```python
for i in range(4, 1, -1):
    print(i)
# affiche
4
3
2
```