

Exercice 1

Suite récurrente et fonction recursive

1.1 Script récursif

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

```

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)

```

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 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, ...

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 récurifs, 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 $n \in \mathbb{N}$.

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

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 ?

```

1 f(99) = f(f(110))
2       = f(100)
3       = f(...
```

Prolongement : Une fonction $x + 1/x$

Considérons la suite u_n

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

définie par

$$u_0 = 1$$

et la relation de récurrence :

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

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

5.2 Complexité

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 récurrence 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 :

```

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

```

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

Exercice 6

Longueur d'une liste

6.1 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. C'est à dire montrer que ce programme va bien retourner la longueur de la chaîne de caractères.

6.2 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'un *slice* python. Quelle est l'instruction python correspondante ?
- 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 :

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

- Prouver la terminaison de l'algorithme récursif.

Exercice 7

Les tours de Hanoï

Voir le cours en ligne sur la complexité. 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 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

```

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 recurrence 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

Dessin recursif

Soit la relation de recurrence suivante :

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

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

8.1 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 .

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

- Représenter le dessin obtenu à l'ordre 2.
- Déterminer expérimentalement la limite de la suite u_n ?

Exercice 9

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.

- Pour la suite, il sera utile de disposer d'une fonction échange 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.

```
1 def échange(lst, i1, i2):
2     lst[i2] = lst[i1]
3     lst[i1] = lst[i2]
```

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

```
1 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)` ?

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

- 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.

```
1 from random import randint
2
3 def melange(lst, ind):
4     print(lst)
5     if ind > 0:
6         j = randint(0, ind)
7         échange(lst, ind, j)
8         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 :

```
1 lst = [v for v in range(5)]
2 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 :

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

```
1 for i in range(4, 1, -1):
2     print(i)
3 # affiche
4 4
5 3
6 2
```

Exercice 10

Dichotomie recursive

Soit `tab` un tableau non vide d'entiers triés dans l'ordre croissant et n un entier.

La fonction `chercher` ci-dessous doit renvoyer un indice où la valeur n apparaît dans `tab` si cette valeur y figure et `None` sinon.

Les paramètres de la fonction sont :

- `tab`, le tableau dans lequel s'effectue la recherche ;
- `x`, l'entier à chercher dans le tableau ;

- i, l'indice de début de la partie du tableau où s'effectue la recherche ;
- j, l'indice de fin de la partie du tableau où s'effectue la recherche.

L'algorithme demandé est une recherche dichotomique récursive.

Recopier et compléter le code de la fonction chercher suivante :

```

1 def chercher(tab, x, i, j):
2     '''Renvoie l'indice de x dans tab, si x est dans tab,
3     None sinon.
4     On suppose que tab est trié dans l'ordre croissant.'''
5     if i > j:
6         return None
7     m = (i + j) // ...
8     if ... < x:
9         return chercher(tab, x, ... , ...)
10    elif tab[m] > x:
11        return chercher(tab, x, ... , ...)
12    else:
13        return ...

```

Exemples :

```

1 >>> chercher([1, 5, 6, 6, 9, 12], 9, 0, 5)
2 4
3 >>> chercher([1, 5, 6, 6, 9, 12], 6, 0, 5)
4 2

```

10.1 Compléter le script

10.2 Que retourne chercher([1, 5, 6, 6, 9, 12], 1, 0, 3) ?

10.3 Que retourne chercher([1, 5, 6, 6, 9, 12], 7, 0, 5) ?

10.4 Faire un tracé de la fonction pour chercher([1, 5, 6, 6, 9, 12], 7, 0, 5) à l'aide d'un tableau :

appel avec i=	j=	m	return
..
..
..
..