

Fiche d'exercices - Chapitre D.1 - La récursivité

Exercice 1

Pour chacune des fonctions suivantes, déterminer s'il s'agit d'une fonction récursive ou non. Si oui, entourer le(s) cas de base.

⚠ Les fonctions 4 et 5 sont liées

1. rebours

```
def rebours(n):  
    print(n)  
    if n > 0:  
        rebours(n-1)  
    else:  
        print("FIN")
```

Fonction réccursive

```
    else:  
        print("FIN")
```

2. u

```
def u(n):  
    res = 5  
    for i in range(n):  
        res = res + n
```

Fonction non réccursive

3. autre_rebours

```
def autre_rebours(n):  
    print(n)  
    if n > 0:  
        autre_rebours(n-1)
```

Fonction réccursive

"else" implicite

4. pair

```
def pair(n):  
    if n == 0:  
        return True  
    elif n == 1:  
        return False  
    else:  
        return impaire(n-1)
```

Réccursif 2 cas de bases

```
if n == 0:  
    return True
```

```
if n == 1:  
    return False
```

5. impair

```
def impair(n):  
    if n==0:  
        return False  
    elif n==1:  
        return True  
    else:  
        return pair(n-1)
```

Réccursif 2 cas de bases

```
if n == 0:  
    return False
```

```
if n == 1:  
    return True
```

Exercice 2

Pour chacune des fonctions précédentes, déterminer un variant de boucle permettant de démontrer que la fonction s'arrête dans tous les cas respectant les conditions d'utilisation.

1. u

```
def u(n):
    """
    :CU: type(n) == int, n > 0
    """
    if n < 1:
        return 1
    else:
        return u(n-1)*2
```

Les valeurs prises par le paramètres n forment une suite d'entier strictement décroissants. On est donc sur d'atteindre le cas $n < 1$ (cas de base).

2. distance

```
def distance(x,y):
    """
    :CU: type(x) == int, type(y) == int, x >= 0, y >= 0
    """
    if x == y:
        return 0
    elif x < y:
        return 1 + distance(x,y-1)
    else:
        return 1 + distance(x-1, y)
```

La valeur absolue de $x - y$ diminue à chaque tour. Donc $x - y$ atteindra la valeur 0 donc $x = y$ (Le cas de base)

3. plus_petit

```
def plus_petit(x,y):
    """
    :CU: type(x) == int, type(y) == int, x >= 0, y >= 0
    """
    if x==y:
        return y
    elif x<=0:
        return x
    else:
        return plus_petit(x-1,y)
```

Les valeurs prises par le paramètres x forment une suite d'entier strictement décroissants. On est donc sur d'atteindre le cas $x \leq 0$ (l'un des cas de base).

Exercice 3

Ecrire une fonction `somme(n)` récursive qui renvoie la somme des n premiers nombres entiers. n étant passé en paramètre.

On s'aidera de la formule suivante :

$$somme(n) = somme(n - 1) + n$$

Exercice 4

Cet exercice est tiré du BAC 2022 - Polynésie - Jour 1

Cet exercice traite du thème «programmation», et principalement de la récursivité.

On s'intéresse dans cet exercice à la construction de chaînes de caractères suivant certaines règles de construction.

Règle A : une chaîne est construite suivant la règle A dans les deux cas suivants:

- soit elle est égale à "a" ;
- soit elle est de la forme "a"+chaîne+"a", où chaîne est une chaîne de caractères construite suivant la règle A.

Règle B : une chaîne est construite suivant la règle B dans les deux cas suivants :

- soit elle est de la forme "b"+chaîne+"b", où chaîne est une chaîne de caractères construite suivant la règle A ;
- soit elle est de la forme "b"+chaîne+"b", où chaîne est une chaîne de caractères construite suivant la règle B.

Documentation de la fonction choice du module random

```
>>>from random import choice
>>>help(choice)
Help on method choice in module random:
choice(seq) method of random.Random instance
Choose a random element from a non-empty sequence.
```

La fonction `A()` ci-dessous renvoie une chaîne de caractères construite suivant la règle A, en choisissant aléatoirement entre les deux cas de figure de cette règle.

```
def A():
    if choice([True, False]):
        return "a"
    else:
        return "a" + A() + "a"
```

1.

(a) Cette fonction est-elle récursive ? Justifier.

(b) La fonction `choice([True, False])` peut renvoyer `False` un très grand nombre de fois consécutives. Expliquer pourquoi ce cas de figure amènerait à une erreur d'exécution.

Dans la suite, on considère une deuxième version de la fonction `A`. À présent, la fonction prend en paramètre un entier `n` tel que, si la valeur de `n` est négative ou nulle, la fonction renvoie `"a"`. Si la valeur de `n` est strictement positive, elle renvoie une chaîne de caractères construite suivant la règle A avec un `n` décrémenté de 1, en choisissant aléatoirement entre les deux cas de figure de cette règle.

```
def A(n):
    if ... or choice([True, False]):
        return "a"
    else:
        return "a" + ... + "a"
```

2.

- (a) Compléter aux emplacements des points de suspension ... le code de cette nouvelle fonction **A**.
- (b) Justifier le fait qu'un appel de la forme **A(n)** avec **n** un nombre entier positif inférieur à 50, termine toujours.
- On donne ci-après le code de la fonction récursive **B** qui prend en paramètre un entier **n** et qui renvoie une chaîne de caractères construite suivant la règle B.

```
def B(n):
    if n <= 0 or choice([True, False]):
        return "b" + A(n-1) + "b"
    else:
        return "b" + B(n-1) + "b"
```

On admet que :

- Les appels **A(-1)** et **A(0)** renvoient la chaîne "a";
- l'appel **A(1)** renvoie la chaîne "a" ou la chaîne "aaa";
- l'appel **A(2)** renvoie la chaîne "a", la chaîne "aaa" ou la chaîne "aaaaa".

3. Donner toutes les chaînes possibles renvoyées par les appels **B(0)**, **B(1)** et **B(2)**.

On suppose maintenant qu'on dispose d'une fonction **raccourcir** qui prend comme paramètre une chaîne de caractères de longueur supérieure ou égale à 2, et renvoie la chaîne de caractères obtenue à partir de la chaîne initiale en lui ôtant le premier et le dernier caractère.

Exemple:

```
>>> raccourcir("abricot")
"brico"
>>> raccourcir("ab")
""
```

4.

- (a) Compléter les points de suspension ... du code de la fonction **regleA** ci-dessous pour qu'elle renvoie **True** si la chaîne passée en paramètre est construite suivant la règle A, et **False** sinon.

```
def regleA(chaine):
    n = len(chaine)
    if n >= 2:
        return chaine[0] == "a" and chaine[n-1] == "a" and regleA(...)
    else:
        return chaine == ...
```

- (b) Écrire le code d'une fonction **regleB**, prenant en paramètre une chaîne de caractères et renvoyant **True** si la chaîne est construite suivant la règle B, et **False** sinon.